The Dissertation Committee for Ioannis Mitliagkas
certifies that this is the approved version of the following dissertation:

Resource-Constrained, Scalable Learning

Committee:

Sriram Vishwanath, Supervisor

Constantine Caramanis, Co-Supervisor

Alexandros Dimakis

Sujay Sanghavi

Pradeep Ravikumar

Resource-Constrained, Scalable Learning

by

Ioannis Mitliagkas, M.Sc.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2015

To my Father

Acknowledgments

The early years of my doctoral work were marked with adjustment difficulties, bad news from home and no lack of mental distress. I owe a tremendous amount to my advisors Constantine and Sriram for their unconditional support and endless patience. Acting in unofficial advisor capacity, Alex Dimakis also offered me his own valuable pieces of advice and endless hours of brainstorming on the board. My undergraduate advisor, Nikos Sidiropoulos, first inspired the love for research in me and provided me with constant support all of these years. I would like to express deep gratitude to all of them; they taught me how good work is done and gave me the confidence to actually do it. Finally, I want to thank my collaborators Aditya, Jubin, Prateek, Dimitris and Michael for working with me; each one taught me something valuable.

On a personal level, I feel indebted to my family. After my parents put me through school in Greece and saw me off to the US, they provided me with emotional and financial support even while they were going through serious adversity back home. My sister, Dina, was there for them in the thick of it, allowing me to pursue my goals abroad.

At a critical turning point, my love, Emily, came and lifted the heavy blanket of depression off of me. She inspired me with confidence and desire to enjoy life and she shared beautiful travels and good food with me. Her energy

motivated me to set higher goals and take on challenges I would not have in the past. Many a time I chased her for 13.1 miles.

I want to thank all of my friends back home for showing a ton of patience and understanding in the face of our highly irregular communication pattern. Also, all of my good friends in Austin: Kumar, Sharayu, Fabio, Aditya, Ankit, Sid, Chinmayi, Dimitris and Megas for sharing good and bad times with me.

Finally, I am grateful to all my musician friends who provided me with a creative outlet both fun and very cathartic. Thank you Arvind, Niko, Priyamvada, Vinit, Constantine, Alex, Tsambika and Taso!

Resource-Constrained, Scalable Learning

by

Ioannis Mitliagkas, Ph.D.
The University of Texas at Austin, 2015

Supervisors: Sriram Vishwanath
Constantine Caramanis

Our unprecedented capacity for data generation and acquisition often reaches the limits of our data storage capabilities. Situations when data are generated faster or at a greater volume than can be stored demand a streaming approach. Memory is an even more valuable resource. Algorithms that use more memory than necessary can pose bottlenecks when processing high-dimensional data and the need for memory-efficient algorithms is especially stressed in the streaming setting. Finally, network along with storage, emerge as the critical bottlenecks in the context of distributed computation. These computational constraints spell out a demand for efficient tools, that guarantee a solution in the face of limited resources, even when the data is very noisy or highly incomplete.

For the first part of this dissertation, we present our work on streaming, memory-limited Principal Component Analysis (PCA). Therein, we give the first convergence guarantees for an algorithm that solves PCA in the single-pass streaming setting. Then, we discuss the distinct challenges that arise when the received samples are overwhelmingly incomplete and present an algorithm and analysis that deals with this issue. Finally, we give a set of extensive experiment results that showcase the practical merits of our algorithm over the state of the art.

The need for heavy network communication arises as the bottleneck when dealing with cluster computation. In that paradigm, a set of worker nodes are connected over the network to produce a cluster with improved computational and storage capacities. This comes with an increased need for communication across the network. In the last part of this work, we consider the problem of PageRank on graph engines. Therein, we make changes to GraphLab, a state-of-the-art platform for distributed graph computation, in a way that leads to a 7x-10x speedup for certain PageRank approximation tasks. Accompanying analysis supports the behaviour we see in our experiments.

Table of Contents

Chapter 1

Introduction

Inexpensive electronics and fast networks lead to data collection capacity that is ever-growing in rate and resolution. Paired with practices of bulk data collection, this trend often meets the limits of data processing and storage. Naturally, computational power is the first resource concern. Poor algorithmic design and implementation can easily lead to a system that is CPU-bound. Beyond the CPU, and depending on the platform, problem, algorithm and implementation, other resources turn out to be the limiting factor: main memory, storage and over-the-network communication are the usual suspects. In this thesis, we study learning problems motivated by a limiting resource. In each case, we identify the limitation, propose an algorithmic/system approach to solving it and support our work with tight analysis and experiments. We now discuss three kinds of resources and then introduce a few problems motivated by them – or lack thereof.

Memory becomes a bottleneck when the prescribed algorithm requires more memory than the application-specific minimum. For example, bad modeling can lead to an unwieldy parameter space, which blows up way past the given system's limitations. Often an algorithmic choice is at the heart of this

issue. Or simply the whole dataset is too big. Consider a problem with an input dataset that exceeds the size of the main memory, but small output size. It would seem reasonable to stream the input data points and only consider them one-by-one, or in small batches. We will see examples like this, where the access mode, algorithm and platform are motivated by these bottlenecks.

Storage becomes the bottleneck when available disks cannot keep up with the rate or total volume of the data acquired. For example, optic fibres outpace most hard disks, including SSDs, and high-definition video recorded by a cell phone would soon demand all of the storage available on the device. In the context of streaming, this further implies that we can only perform a *single pass* over the whole dataset, as we cannot store it in its entirety. In the single-pass streaming paradigm, each sample collected is looked at once, any useful information is extracted and then the sample is discarded. For the first part of our work, we focus on streaming, memory-limited problems constrained by these memory and storage bottlenecks.

Network becomes the bottleneck for distributed systems. Computation is balanced across many machines in a cluster to leverage much more computational power than that of a single machine. This computational gain comes at the cost of increased communication needs. For the latter part of our work, we turn our attention to the distributed approximation of PageRank, one of the most important graph analytics tasks. We provide a new system that outperforms the state of the art, along with supporting analysis.

1.1 Streaming Principal Component Analysis

Principal component analysis (PCA) is a fundamental tool for dimensionality reduction, clustering, classification, and many more learning tasks. It is a basic preprocessing step for learning, recognition, and estimation procedures. The core computational element of PCA is performing a (partial) singular value decomposition, and much work over the last half century has focused on efficient algorithms (e.g., [29] and references therein) and hence on *computational complexity*.

The recent focus on understanding high-dimensional data, where the dimensionality of the data scales together with the number of available sample points, has led to an exploration of the *sample complexity* of covariance estimation. This direction was largely influenced by Johnstone's *spiked covariance model*, where data samples are drawn from a distribution whose (population) covariance is a low-rank perturbation of the identity matrix [39]. Work initiated there, and also work done in [72] (and references therein) has explored the power of batch PCA in the p-dimensional setting with sub-Gaussian noise, and demonstrated that the singular value decomposition (SVD) of the empirical covariance matrix succeeds in recovering the principal components (extreme eigenvectors of the population covariance) with high probability, given $n = O(p)$ samples.

The first part of our work brings the focus on another critical quantity: memory/storage. The only currently available algorithms with provable sample complexity guarantees either store all $n = O(p)$ samples (note that

for more than a single pass over the data, the samples must all be stored) or explicitly form or approximate the empirical $p \times p$ (typically dense) covariance matrix. All cases require as much as $O(p^2)$ storage for exact recovery. In certain high-dimensional applications, where data points are high resolution photographs, biometrics, video, etc., p often is of the order of $10^{10} - 10^{12}$, making the need for $O(p^2)$ memory prohibitive. At many computing scales, manipulating vectors of length $O(p)$ is possible, when storage of $O(p^2)$ is not. A typical desktop may have 10-20 GB of RAM, but will not have more than a few TB of total storage. A modern smart-phone may have as much as a GB of RAM, but has a few GB, not TB, of storage. In distributed storage systems, the scalability in storage comes at the heavy cost of communication.

In this light, we consider the *streaming data* setting, where the samples $\mathbf{x}_t \in \mathbb{R}^p$ are collected sequentially, and unless we store them, they are irretrievably gone.[1] For the *spiked covariance model* (and natural generalizations), we show that a simple algorithm requiring $O(kp)$ storage – the best possible – performs as well as batch algorithms (namely, SVD on the empirical covariance matrix), with sample complexity $O(p \log p)$. We discuss related work in Chapter 2 and present our results in detail in Chapter 3. To the best of our knowledge, this is the first algorithm with both storage complexity and sample complexity guarantees.

[1] This is similar to what is sometimes referred to as the *single-pass* model.

1.2 Missing Entries

The second part of our work, considers the problem of PCA under *severe memory/storage constraints and partial observation* – a setting where to the best of our knowledge, there are no known algorithms with global performance guarantees. Again, we consider the streaming setting, where we see data points sequentially, and these are nowhere stored. We seek to use *no more total storage than required for the output*. This essentially means that while our algorithm sees each data point, it can only do so once.

Motivated by many recent applications in preference and behavior modeling, we focus on the partial observation setting: each data point is not only noisy as in traditional PCA. It also suffers some – perhaps overwhelming – number of erasures. Extreme erasures are typical of application areas where the data are naturally sparse. Moreover, erasures may be an introduced feature, e.g. where features are withheld to protect privacy, while allowing collective learning. It is possible to erase features in a way that makes it impossible to complete any one sample, while principal components can still be extracted. This distinction between extracting principle components and *completing* the data, becomes important in our work as it sets it apart from imputation-based methods (cf. Section 2.4).

We consider the following problem: given partial observations $\dot{\mathbf{x}}_t$ of vectors $\mathbf{x}_t \in \mathbb{R}^p$, we seek a k-dimensional subspace U along which the variance of complete vectors $\{\mathbf{x}_i\}$ is maximized. Matrix completion techniques using either SVD [41] or nuclear norm optimization approaches (e.g., [22, 53, 60, 43])

have formed the bulk of research into such problems. Largely, their algorithmic and statistical performance is quite well understood. Yet these algorithms all have storage complexity on the order of $O(p^2)$. Moreover, most of these results focus on *matrix completion* – an objective of less value in the setting where we cannot store points.

In contrast, we focus on recovering a subspace close to U. If there are enough observations, this is equivalent to matrix completion, but that is not always the case. Recent work on memory-restricted PCA has considered this objective, including [51, 6, 9, 11]. Of these, our work in [51] is the only one that guarantees optimal sample complexity and global convergence, yet does not consider erasures; [11, 12] is the only one that can handle erasures, though convergence guarantees there are only local. The algorithms in [11] (GROUSE) and [55] (Stochastic Approximation, finally analyzed in [9]) perform very well in general but they share two important drawbacks: i) their performance can suffer when the number of erasures is very large and ii) their success critically depends on careful individual parametrization for every dataset. These are discussed in more detail in Sections 2.2 and 4.4.

Our contributions: We provide a practical, easy-to-deploy algorithm that does not suffer from the above issues. To the best of our knowledge, it is the first algorithm for streaming PCA with erasures, that comes with global convergence guarantees. Specifically:

- **Algorithm and Performance**: We provide a simple, fast algorithm

that has the form of a block power method update. We experiment on synthetic and real data, and demonstrate the performance of our algorithm.

- **Implementation and deployability**: Perhaps its most salient quality is the fact that it obviates the need for "guesswork" when deploying on a new dataset. That is, unlike other streaming algorithms, the same exact parametrization, can perform well in different datasets. See Section 4.4 for the numbers and discussion.

- **Sample Complexity**: Under mild assumptions on the data distribution, we show that our algorithm recovers the k principal components with $\tilde{O}(p/\delta^2\epsilon^2)$ samples, which is scaling-wise optimal for any algorithm. Furthermore, we show – in theory and experiments – that we can recover U even when $\delta p < k$. In this regime, *matrix completion is generally not possible*.

- **Memory Complexity**: Our algorithm requires memory $O(pk)$ – this is the best possible. This much memory is required to store the output alone.

We discuss connections to past work in Chapter 2 and introduce the system model in detail in Section 4.1. We then present our algorithm and analysis in Section 4.2, and Section 4.3, respectively, and conclude with extensive experiments in Section 4.4.

1.3 PageRank Approximations on Very Large Graphs

For the last part of this dissertation, we turn our attention to a different computational paradigm. Large-scale graph processing is becoming increasingly important for the analysis of data from social networks, web pages, bioinformatics and recommendation systems. Graph algorithms are difficult to implement in distributed computation frameworks like Hadoop MapReduce and Spark. For this reason several in-memory graph engines like Pregel, Giraph, GraphLab and GraphX [49, 48, 77, 65] are being developed.

PageRank computation [56], which gives an estimate of the importance of each vertex in the graph, is a core component of many search routines; more generally, it represents, de facto, one of the canonical tasks performed using such graph processing frameworks. Indeed, while important in its own right, it also represents the memory, computation and communication challenges to be overcome in large scale iterative graph algorithms.

In this dissertation we propose a novel algorithm for fast *approximate calculation* of high PageRank vertices. Note that even though most previous works calculate the complete PageRank vector (of length in the millions or billions), in many graph analytics scenarios a user wants a quick estimation of the most important or relevant nodes – distinguishing the 10^{th} most relevant node from the $1\,000^{th}$ most relevant is important; the $1\,000\,000^{th}$ from the $1\,001\,000^{th}$ much less so. A simple solution is to run the standard PageRank algorithm for fewer iterations (or with an increased tolerance). While certainly incurring less overall cost, the per-iteration cost remains the same;

more generally, the question remains whether there is a more efficient way to approximately recover the heaviest PageRank vertices.

There are many real-world applications that may benefit from a fast top-k PageRank algorithm. One example is *growing loyalty of influential customers* [1]. In this application, a telecom company identifies the top-k influential customers using the top-k PageRank on the customers' activity (e.g., calls) graph. Then, the company invests its limited budget on improving user experience for these top-k customers, since they are most important for building good reputation. Another interesting example is finding keywords and key sentences in a given text. In [50], the authors show that PageRank performs better than known machine learning techniques for keyword extraction. Each unique word (noun, verb or an adjective) is regarded as a vertex and there is an edge between two words if they occur in close proximity in the text. Using approximate top-k PageRank, we can identify the top-k keywords much faster than obtaining the full ranking. When keyword extraction is used by time sensitive applications or for an ongoing analysis of a large number of documents, speed becomes a crucial factor. The last example we describe here is the application of PageRank for online social networks (OSN). It is important in the context of OSNs to be able to predict which users will remain active in the network for a long time. Such *key users* play a decisive role in developing effective advertising strategies and sophisticated customer loyalty programs, both vital for generating revenue [37]. Moreover, the remaining users can be leveraged, for instance for targeted marketing or premium ser-

vices. It is shown in [37] that PageRank is a much more efficient predictive measure than other centrality measures. The main innovation of [37] is the usage of a mixture of connectivity and activity graphs for PageRank calculation. Since these graphs are highly dynamic (especially the user activity graph), PageRank should be recalculated constantly. Moreover, the *key users* constitute only a small fraction of the total number of users, thus, a fast approximation for the top-PageRank nodes constitutes a desirable alternative to the exact solution.

In Chapter 5 of this dissertation we address this problem. Our algorithm (called FrogWild! for reasons that will become subsequently apparent) significantly outperforms the simple reduced iterations heuristic in terms of *running time, network communication* and exhibits better scaling. We note that, naturally, we compare our algorithm and reduced-iteration-PageRank within the same framework: we implemented our algorithm in GraphLab PowerGraph and compare it against the built-in PageRank implementation. A key part of our contribution also involves the proposal of what appears to be simply a technically minor modification within the GraphLab framework, but nevertheless results in significant network-traffic savings, and we believe may nevertheless be of more general interest beyond PageRank computations.

Chapter 2

Background

2.1 Notation

Lowercase letters denote scalars or vectors. Uppercase letters denote matrices. $\|\mathbf{x}\|_q$ denotes the ℓ_q norm of $\mathbf{x}$; $\|\mathbf{x}\|$ denotes the ℓ_2 norm of $\mathbf{x}$. $\|A\|$ or $\|A\|_2$ denotes the spectral norm of A while $\|A\|_F$ denotes the Frobenius norm of A. If U is a matrix, $U_\perp$ denotes an orthogonal basis for the subspace perpendicular to span(U). We denote its ith row by U^i and its ith column by $\mathbf{u}_i$. Similarly for a vector $\mathbf{x}$, we write x^i for its ith entry. For $\mathbf{x} \in \mathbb{R}^p$ and $\Omega \subseteq [p]$, we use $\mathbf{x}_\Omega$ to denote the restriction of $\mathbf{x}$ to the elements in the set Ω. Finally, we write $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^\top \mathbf{b}$ for the inner product between $\mathbf{a}$, $\mathbf{b}$. The (i, j) element of a matrix A is A_{ij}. We denote the transpose of a matrix A by A'. We use Δ^{n-1} for the probability simplex in n dimensions, and and $e_i \in \Delta^{n-1}$ for the indicator vector for item i. For example, $e_1 = [1, 0, ...0]$. For the set of all integers from 1 to n we write $[n]$.

2.2 Streaming PCA

Memory- and computation-efficient algorithms that operate on streaming data are plentiful in the literature and many seem to do well in practice.

While much work has focused on memory-constrained PCA, there has as of yet been no work that simultaneously provides sample complexity guarantees competitive with batch algorithms, and also memory/storage complexity guarantees close to the minimal requirement of $O(kp)$ – the memory required to store only the output (cf. Section 3.2 for a listing of prior work). Because of the practical relevance, there is renewed interest in this problem with various sources mentioning it as an important unresolved issue (e.g., [75, 5]). In Chapter 3, we present an algorithm that provably meets both objectives stated above.

For the partially observed setting, two directions stand out: covariance estimation and imputation-based algorithms. The motivating discussion in Section 1.2 contains a list of references for them, and the next two sections provide detailed insight into their strengths and weaknesses.

2.3 Unbiased Covariance Estimation

A critical element of many PCA algorithms is some form of covariance estimation, be that explicit or implicit. The former is true for the classic batch PCA algorithm. The algorithm computes the empirical covariance matrix,

$$\Sigma_n = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T, \tag{2.1}$$

and then performs a Singular Value Decomposition (SVD) on Σ_n to recover the range of U. The statistical limits of this process are characterized in [40]. Specifically, $O(p)$ samples are necessary for the recovery of U in the full-rank,

subgaussian case. This includes the spiked covariance model.

The introduction of erasures in the data stream, renders the estimator in (2.1) biased. The authors in [47] discuss this issue for the batch setting and provide an alternative algorithm. It is based on,

$$\tilde{\Sigma}_n = \delta^{-2}\Sigma_n + (\delta^{-1} - \delta^{-2})\text{diag}(\Sigma_n), \tag{2.2}$$

and employs regularized optimization to make the method efficient in the high-dimensional case. That algorithm is not applicable in a streaming setup, however the estimator in (2.2) and accompanying concentration analysis, provided therein is a useful tool for our purposes.

2.4 Imputation-based Algorithms

A line of empirically successful algorithms introduced in [11] (GROUSE) and studied further in [35] and [12], avoid covariance estimation. To that end, they use updates that resemble stochastic approximation, except they are performed along the Grassmanian manifold. In its general form, the algorithm first calculates the projection of the latest sample, $\dot{\mathbf{x}}_t$, on the current subspace estimate, say Q_t. As per the model, only a subset Ω_t of indices is observed, i.e. $\dot{\mathbf{x}}_t|_{\Omega_t} = \mathbf{x}_t|_{\Omega_t}$ and $\dot{\mathbf{x}}_t|_{\overline{\Omega}_t} = 0$. Restricting $\dot{\mathbf{x}}_t$ and Q_t to the observed indices, the projection is calculated as follows:

$$\mathbf{w}_t = \text{argmin}_{\mathbf{w}\in\mathbb{R}^k} \left\| \dot{\mathbf{x}}_t|_{\Omega_t} - Q_t|_{\Omega_t}\mathbf{w} \right\|_2. \tag{2.3}$$

Then, Q_t and the optimal weights in $\mathbf{w}_t$ are used to impute the entries missing from $\dot{\mathbf{x}}_t$.

$$\dot{\mathbf{x}}_t|_{\overline{\Omega}_t} \leftarrow Q_t|_{\overline{\Omega}_t}\mathbf{w}_t \tag{2.4}$$

Finally, the algorithm uses the imputed vector to update Q_t, performing a descent step on the Grassmanian.

This method has proven to perform well in practice. However, in the regime where the number of observed elements per vector ($|\Omega_t|$) is less than the number of components (k), the projection in (2.3) is underdetermined, making the step ill-defined. Picking the minimum-norm solution is a reasonable way to deal with this issue and we put this idea to the test in our experiments. The results in Section 4.4, suggest that even though this "fix" makes the imputation step well-defined, it does not lead to accurate recovery of the ground truth.

Another natural way to modify these algorithms to deal with this case, is discarding all samples with an insufficient number of observed entries (less than k). This makes a very small difference in experiments – not included here for brevity – but there is a simple probabilistic argument against it: Assuming each entry is observed independently, the number of observed entries is given by a binomial random variable (more generally Poisson trials). For $k = (c+1)\delta p$, with $c > 0$, a Chernoff bound gives

$$\mathbb{P}\left(|\Omega_t| \geq k\right) \leq \exp\left\{-\left(c^2 \wedge c\right)\delta p/3\right\}. \tag{2.5}$$

This implies that, for any $c > 0$, the number of wasted samples would range from large to overwhelming, depending on the scaling of δp.

We conclude that methods based on projection-based imputation face significant problems in the regime of many missing entries and provide an alternative in Chapter 4.

2.5 Large-scale Graph Computation

Graph engines provide simple programming abstractions and take care of partitioning and storing very large graphs and their metadata on the machines of a cluster. This partitioning makes it possible to deal with graphs that are otherwise too big for a single machine. Furthermore, the total computational load can be balanced across the cluster. A number of different graph engines are under active development. Pregel, Giraph, GraphLab and GraphX [49, 48, 77, 65] each provide slightly different functionality and abstractions. There is no full consensus on the fundamental abstractions of graph processing frameworks but certain patterns such as vertex programming and the Bulk Synchronous Parallel (BSP) framework seem to be increasingly popular.

Vertex programming refers to a class of programming abstractions. The programmer essentially gives instructions to a vertex, in the form of a message-passing-like scheme. In the abstraction of Pregel – also adopted by GraphLab and other engines – the life of a vertex is split into three stages:

Gather The vertex wakes up and collects messages from its *immediate neighbourhood*.

Apply The vertex performs some computation based on its current state and

the received messages and calculates its new state.

Scatter The vertex sends messages to its immediate neighbourhood.

This abstraction is surprisingly powerful and expressive, but most importantly, it allows the engine to perform a number of performance optimization steps under the hood (cf. [30]). The engine also takes care of vertex scheduling, which can be synchronous or asynchronous. The most commonly used scheme is Bulk Synchronous Parallel [70]: the stages of gather/apply/scatter are grouped into *super-steps*. Using global synchronization, the engine schedules a subset of all vertices and only considers a super-step finished when all running vertices are finished.

2.6 PageRank

Consider a directed graph $G = (V, E)$ with n vertices ($|V| = n$) and let A denote its adjacency matrix. That is, $A_{ij} = 1$ if there is an edge from j to i. Otherwise, the value is 0. Let $d_{\text{out}}(j)$ denote the number of *successors* (out-degree) of vertex j in the graph. We assume that all nodes have at least one successor, $d_{\text{out}}(j) > 0$. Then we can define the transition probability matrix P as follows:

$$P_{ij} = A_{ij}/d_{\text{out}}(j). \tag{2.6}$$

The matrix is left-stochastic, which means that each of its columns sums to 1. We call $G(V, E)$ the *original graph*, as opposed to the PageRank graph,

which includes a weighted edge between any pair of vertices. We now define its transition probability matrix, and the PageRank vector.

Definition 1 (PageRank [56])**.** *Consider the matrix*

$$Q \triangleq (1 - p_T)P + p_T \frac{1}{n} 1_{n \times n}.$$

where $p_T \in [0, 1]$ *is a parameter, most commonly set to* 0.15*. The PageRank vector* $\pi \in \Delta^{n-1}$ *is defined as the principal right eigenvector of* Q*. That is,* $\pi \triangleq v_1(Q)$*. By the Perron-Frobenius theorem, the corresponding eigenvalue is* 1*. This implies the* fixed-point *characterization of the PageRank vector,* $\pi = Q\pi$*.*

The PageRank vector assigns high values to *important* nodes. Intuitively, important nodes have many important predecessors (other nodes that point to them). This recursive definition is what makes PageRank robust to manipulation, but also expensive to compute. It can be recovered by exact eigendecomposition of Q, but at real problem scales this is prohibitively expensive. In practice, engineers often use a few iterations of the power method to get a "good-enough" approximation.

The definition of PageRank hinges on the left-stochastic matrix Q, suggesting a connection to Markov chains. Indeed, this connection is well documented and studied [2, 28]. An important property of PageRank from its random walk characterization, is the fact that π is the invariant distribution for a Markov chain with dynamics described by Q. A non-zero p_T, also called

the *teleportation probability*, introduces a uniform component to the PageRank vector π. We see in our analysis that this implies ergodicity and faster mixing for the random walk.

Chapter 3

Memory-constrained, Streaming PCA

In this chapter, we formally pose the problem, as introduced in Section 1.1. Then we propose an algorithm to solve it and give theoretical guarantees on its convergence in Sections 3.3.1 and 3.3.2. The experiments in Section 3.6 support our theoretical findings.

3.1 Problem Formulation

We consider the streaming model: at each time step t, we receive a point $\mathbf{x}_t \in \mathbb{R}^p$. Any point that is not explicitly stored can never be revisited. Our goal is to compute the top k principal components of the data: the k-dimensional subspace that offers the best squared-error estimate for the points. We assume a probabilistic generative model, from which the data is sampled at each step t. Specifically,

$$\mathbf{x}_t = A\mathbf{z}_t + \mathbf{w}_t, \tag{3.1}$$

where $A \in \mathbb{R}^{p\times k}$ is a fixed matrix, $\mathbf{z}_t \in \mathbb{R}^{k\times 1}$ is a multivariate normal random variable, i.e.,

$$\mathbf{z}_t \sim \mathcal{N}(0_{k\times 1}, I_{k\times k}),$$

and $\mathbf{w}_t \in \mathbb{R}^{p\times 1}$ is the "noise" vector, also sampled from a multivariate normal distribution, i.e.,

$$\mathbf{w}_t \sim \mathcal{N}(0_{p\times 1}, \sigma^2 I_{p\times p}).$$

Furthermore, we assume that all $2n$ random vectors $(\mathbf{z}_t, \mathbf{w}_t, \forall 1 \leq t \leq n)$ are mutually independent.

In this regime, it is well-known that batch-PCA is asymptotically consistent (hence recovering A up to unitary transformations) with number of samples scaling as $n = O(p)$ [73]. It is interesting to note that in this high-dimensional regime, the signal-to-noise ratio quickly approaches zero, as the signal, or "elongation" of the major axis, $\|Az\|_2$, is $O(1)$, while the noise magnitude, $\|\mathbf{w}\|_2$, scales as $O(\sqrt{p})$. The central goal of this work is to provide finite sample guarantees for a streaming algorithm that requires memory no more than $O(kp)$ and matches the consistency results of batch PCA in the sampling regime $n = O(p)$ (possibly with additional log factors, or factors depending on σ and k).

3.2 Prior Work

Online-PCA for *regret minimization* is considered in several papers, most recently in [75]. There, the multiplicative weights approach is adapted to this problem, with experts corresponding to subspaces. The goal is to control the regret, improving on the natural follow-the-leader algorithm that performs batch-PCA at each step. However, the algorithm can require $O(p^2)$ memory, in order to store the multiplicative weights. A memory-light variant described

in [5] typically requires much less memory, but there are no guarantees for this, and moreover, for certain problem instances, its memory requirement is on the order of p^2.

Sub-sampling, dimensionality-reduction and sketching form another family of low-complexity and low-memory techniques, see, e.g., [23, 52, 32]. These save on memory and computation by performing SVD on the resulting smaller matrix. The results in this line of work provide worst-case guarantees over the pool of data, and typically require a rapidly decaying spectrum, not required in our setting, to produce good bounds. More fundamentally, these approaches are not appropriate for data coming from a statistical model such as the spiked covariance model. It is clear that subsampling approaches, for instance, simply correspond to discarding most of the data, and for fundamental sample complexity reasons, cannot work. Sketching produces a similar effect: each column of the sketch is a random $(+/-)$ sum of the data points. If the data points are, e.g., independent Gaussian vectors, then so will each element of the sketch, and thus this approach again runs against fundamental sample complexity constraints. Indeed, it is straightforward to check that the guarantees presented in ([23, 32]) are not strong enough to guarantee recovery of the spike. This is not because the results are weak; it is because they are geared towards worst-case bounds.

Algorithms focused on sequential SVD (e.g., [19, 18], [24],[45] and more recently [10, 34]) seek to have the best subspace estimate at every time (i.e., each time a new data sample arrives) but without performing full-blown SVD

at each step. While these algorithms indeed reduce both the computational and memory burden of batch-PCA, there are no rigorous guarantees on the quality of the principal components or on the *statistical performance* of these methods.

In a Bayesian mindset, some researchers have come up with expectation maximization approaches [62, 68], that can be used in an incremental fashion. The finite sample behavior is not known.

Stochastic-approximation-based algorithms along the lines of [61] are also quite popular, due to their low computational and memory complexity, and excellent empirical performance. They go under a variety of names, including *Incremental PCA* (though the term *Incremental* has been used in the online setting as well [38]), Hebbian learning [36], and stochastic power method [5]. The basic algorithms are some version of the following: upon receiving data point $\mathbf{x}_t$ at time t, update the estimate of the top k principal components via:

$$U^{(t+1)} = \text{Proj}(U^{(t)} + \eta_t \mathbf{x}_t \mathbf{x}_t^\top U^{(t)}), \tag{3.2}$$

where $\text{Proj}(\cdot)$ denotes the "projection" that takes the SVD of the argument, and sets the top k singular values to 1 and the rest to zero (see [5] for discussion). Though this kind of algorithm performs well empirically, the best known analysis ([9]) guarantees an $\Omega(p^2)$ sample complexity, when $k = 1$. This is an order of magnitude greater compared to the necessary and sufficient $O(p)$ given in [40].

Another important line of work, is the Expectation - Maximization (EM) approach [26]. Unfortunately, there are no global guarantees for EM in this case, nor is it clear how the M-step would be implemented without violating the memory constraint.

In summary, while much work has focused on memory-constrained PCA, there has as of yet been no work that simultaneously provides sample complexity guarantees competitive with batch algorithms, and also memory/storage complexity guarantees close to the minimal requirement of $O(kp)$ – the memory required to store only the output. In Chapter 3, we present an algorithm that provably does both.

3.3 Algorithm and Guarantees

In this section, we present our proposed algorithm and its finite sample analysis. It is a block-wise stochastic variant of the classical power-method. Stochastic versions of the power method already exist in the literature; see [5]. The main impediment to the analysis of such stochastic algorithms (as in (3.2)) is the large variance of each step, in the presence of noise. This motivates us to consider a modified stochastic power method algorithm, that has a variance reduction step built in. At a high level, our method updates only once in a "block" and within one block we average out noise to reduce the variance.

Below, we first illustrate the main ideas of our method as well as our sample complexity proof for the simpler rank-1 case. The rank-1 and rank-

Algorithm 1

Block-Stochastic Power Method	Block-Stochastic Orthogonal Iteration
input $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, Block size: B	
$\mathbf{q}_0 \sim \mathcal{N}(0, I_{p\times p})$ (Initialization)	$H^i \sim \mathcal{N}(0, I_{p\times p}), 1 \le i \le k$ (Initialization)
$\mathbf{q}_0 \leftarrow \mathbf{q}_0/\|\mathbf{q}_0\|_2$	$H \leftarrow Q_0 R_0$ (QR-decomposition)
for $\tau = 0, \ldots, n/B - 1$ **do**	
$\quad \mathbf{s}_{\tau+1} \leftarrow 0$	$S_{\tau+1} \leftarrow 0$
$\quad$ **for** $t = B\tau + 1, \ldots, B(\tau+1)$ **do**	
$\quad\quad \mathbf{s}_{\tau+1} \leftarrow \mathbf{s}_{\tau+1} + \frac{1}{B}\langle \mathbf{q}_\tau, \mathbf{x}_t\rangle \mathbf{x}_t$	$S_{\tau+1} \leftarrow S_{\tau+1} + \frac{1}{B}\mathbf{x}_t\mathbf{x}_t^\top Q_\tau$
$\quad$ **end for**	
$\quad \mathbf{q}_{\tau+1} \leftarrow \mathbf{s}_{\tau+1}/\|\mathbf{s}_{\tau+1}\|_2$	$S_{\tau+1} = Q_{\tau+1}R_{\tau+1}$ (QR-decomposition)
end for	
output	

k algorithms are so similar, that we present them in the same panel. We provide the rank-k analysis in Section 3.3.2. We note that, while our algorithm describes $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ as "input," we mean this in the streaming sense: the data are no-where stored, and can never be revisited unless the algorithm explicitly stores them.

3.3.1 Rank-One Case

We first consider the rank-1 case for which each sample $\mathbf{x}_t$ is generated using: $\mathbf{x}_t = \mathbf{u}\mathbf{z}_t + \mathbf{w}_t$ where $\mathbf{u} \in \mathbb{R}^p$ is the principal component that we wish to recover. Our algorithm is a block-wise method where all the n samples are divided in n/B blocks (for simplicity we assume that n/B is an integer). In the $(\tau+1)$-st block, we compute

$$\mathbf{s}_{\tau+1} = \left(\frac{1}{B}\sum_{t=B\tau+1}^{B(\tau+1)} \mathbf{x}_t\mathbf{x}_t^\top\right)\mathbf{q}_\tau. \tag{3.3}$$

Then, the iterate $\mathbf{q}_\tau$ is updated using $\mathbf{q}_{\tau+1} = \mathbf{s}_{\tau+1}/\|\mathbf{s}_{\tau+1}\|_2$. Note that, $\mathbf{s}_{\tau+1}$

can be computed online, with $O(p)$ operations per step. Furthermore, storage requirement is also linear in p.

3.3.1.1 Analysis

We now present the sample complexity analysis of our proposed method. Using $O(\sigma^4 p \log(p)/\epsilon^2)$ samples, Algorithm 1 obtains a solution $\mathbf{q}_T$ of accuracy ϵ, i.e. $\|\mathbf{q}_T - \mathbf{u}\|_2 \leq \epsilon$.

Theorem 2. *Denote the data stream by $\mathbf{x}_1, \ldots, \mathbf{x}_n$, where $\mathbf{x}_t \in \mathbb{R}^p, \forall t$ is generated by (3.1). Set the total number of iterations $T = \Omega(\frac{\log(p/\epsilon)}{\log((\sigma^2+.75)/(\sigma^2+.5))})$ and the block size $B = \Omega(\frac{(1+3(\sigma+\sigma^2)\sqrt{p})^2 \log(T)}{\epsilon^2})$. Then, with probability 0.99, $\|\mathbf{q}_T - \mathbf{u}\|_2 \leq \epsilon$, where $\mathbf{q}_T$ is the T-th iterate of Algorithm 1. That is, Algorithm 1 obtains an ϵ-accurate solution with number of samples (n) given by:*

$$n = \tilde{\Omega}\left(\frac{(1+3(\sigma+\sigma^2)\sqrt{p})^2 \log(p/\epsilon)}{\epsilon^2 \log((\sigma^2+.75)/(\sigma^2+.5))}\right).$$

Note that in the total sample complexity, we use the notation $\tilde{\Omega}(\cdot)$ to suppress the extra $\log(T)$ factor for clarity of exposition, as T already appears in the expression linearly.

Proof. The proof decomposes the current iterate into the component of the current iterate, $\mathbf{q}_\tau$, in the direction of the true principal component (the spike) $\mathbf{u}$, and the perpendicular component, showing that the former eventually dominates. Doing so hinges on three key components: (a) for large enough B, the empirical covariance matrix $F_{\tau+1} = \frac{1}{B}\sum_{t=B\tau+1}^{B(\tau+1)} \mathbf{x}_t \mathbf{x}_t^\top$ is close to the true

covariance matrix $M = \mathbf{u}\mathbf{u}^\top + \sigma^2 I$, i.e., $\|F_{\tau+1} - M\|_2$ is small. In the process, we obtain "tighter" bounds for $\|\mathbf{u}^\top(F_{\tau+1} - M)\mathbf{u}\|$ for *fixed* $\mathbf{u}$; (b) with probability 0.99 (or any other constant probability), the initial point $\mathbf{q}_0$ has a component of at least $O(1/\sqrt{p})$ magnitude along the true direction $\mathbf{u}$; (c) after τ iterations, the error in estimation is at most $O(\gamma^\tau)$ where $\gamma < 1$ is a constant.

There are several results that we use repeatedly, which we collect here, and prove individually in Section 3.4.

Lemma 3. *Let B, T and the data stream $\{\mathbf{x}_t\}$ be as defined in Theorem 2. Then, w.p. $1 - C/T$ we have:*

$$\left\| \frac{1}{B} \sum_t \mathbf{x}_t \mathbf{x}_t^\top - \mathbf{u}\mathbf{u}^\top - \sigma^2 I \right\|_2 \leq \epsilon.$$

Lemma 4. *Let B, T and the data stream $\{\mathbf{x}_t\}$ be as defined in Theorem 2. Then, w.p. $1 - C/T$ we have:*

$$\mathbf{u}^\top \mathbf{s}_{\tau+1} \geq \mathbf{u}^\top \mathbf{q}_\tau (1 + \sigma^2) \left(1 - \frac{\epsilon}{4(1+\sigma^2)} \right),$$

where $\mathbf{s}_t = \frac{1}{B} \sum_{B\tau < t \leq B(\tau+1)} \mathbf{x}_t \mathbf{x}_t^\top \mathbf{q}_\tau$.

Lemma 5. *Let $\mathbf{q}_0$ be the initial guess for $\mathbf{u}$, given by Steps 1 and 2 of Algorithm 1. Then, w.p. 0.99: $|\langle \mathbf{q}_0, \mathbf{u} \rangle| \geq \frac{C_0}{\sqrt{p}}$, where $C_0 > 0$ is a universal constant.*

Step (a) is proved in Lemmas 3 and 4, while Lemma 5 provides the required result for the initial vector $\mathbf{q}_0$. Using these lemmas, we next complete

the proof of the theorem. We note that both (a) and (b) follow from well-known results; we provide them for completeness.

Let $\mathbf{q}_\tau = \sqrt{1-\delta_\tau}\mathbf{u} + \sqrt{\delta_\tau}\mathbf{g}_\tau, 1 \le \tau \le n/B$, where $\mathbf{g}_\tau$ is the component of $\mathbf{q}_\tau$ that is perpendicular to $\mathbf{u}$ and $\sqrt{1-\delta_\tau}$ is the magnitude of the component of $\mathbf{q}_\tau$ along $\mathbf{u}$. Note that $\mathbf{g}_\tau$ may well change at each iteration; we only wish to show $\delta_\tau \to 0$.

Now, using Lemma 4, the following holds with probability at least $1 - C/T$:

$$\mathbf{u}^\top \mathbf{s}_{\tau+1} \ge \sqrt{1-\delta_\tau}(1+\sigma^2)\left(1 - \frac{\epsilon}{4(1+\sigma^2)}\right). \tag{3.4}$$

Next, we consider the component of $\mathbf{s}_{\tau+1}$ that is perpendicular to $\mathbf{u}$:

$$\mathbf{g}_{\tau+1}^\top \mathbf{s}_{\tau+1} = \mathbf{g}_{\tau+1}^\top \left(\frac{1}{B}\sum_{t=B\tau+1}^{B(\tau+1)} \mathbf{x}_t\mathbf{x}_t^\top\right)\mathbf{q}_\tau = \mathbf{g}_{\tau+1}^\top (M + E_\tau)\mathbf{q}_\tau,$$

where $M = \mathbf{u}\mathbf{u}^\top + \sigma^2 I$ and E_τ is the error matrix: $E_\tau = M - \frac{1}{B}\sum_{t=B\tau+1}^{B(\tau+1)} \mathbf{x}_t\mathbf{x}_t^\top$. Using Lemma 3, $\|E_\tau\|_2 \le \epsilon$ (w.p. $\ge 1 - C/T$). Hence, w.p. $\ge 1 - C/T$:

$$\mathbf{g}_{\tau+1}^\top \mathbf{s}_{\tau+1} = \sigma^2 \mathbf{g}_{\tau+1}^\top \mathbf{q}_\tau + \|\mathbf{g}_{\tau+1}\|_2 \|E_\tau\|_2 \|\mathbf{q}_\tau\|_2 \le \sigma^2\sqrt{\delta_\tau} + \epsilon. \tag{3.5}$$

Now, since $\mathbf{q}_{\tau+1} = \mathbf{s}_{\tau+1}/\|\mathbf{s}_{\tau+1}\|_2$,

$$\begin{aligned}
\delta_{\tau+1} = (\mathbf{g}_{\tau+1}^\top \mathbf{q}_{\tau+1})^2 &= \frac{(\mathbf{g}_{\tau+1}^\top \mathbf{s}_{\tau+1})^2}{(\mathbf{u}^\top \mathbf{s}_{\tau+1})^2 + (\mathbf{g}_{\tau+1}^\top \mathbf{s}_{\tau+1})^2}, \\
&\overset{(i)}{\le} \frac{(\mathbf{g}_{\tau+1}^\top \mathbf{s}_{\tau+1})^2}{(1-\delta_\tau)\left(1+\sigma^2-\frac{\epsilon}{4}\right)^2 + (\mathbf{g}_{\tau+1}^\top \mathbf{s}_{\tau+1})^2}, \\
&\overset{(ii)}{\le} \frac{(\sigma^2\sqrt{\delta_\tau}+\epsilon)^2}{(1-\delta_\tau)\left(1+\sigma^2-\frac{\epsilon}{4}\right)^2 + (\sigma^2\sqrt{\delta_\tau}+\epsilon)^2},
\end{aligned} \tag{3.6}$$

where, (i) follows from (3.4) and (ii) follows from (3.5) along with the fact that $\frac{x}{c+x}$ is an increasing function in x for $c, x \geq 0$. Assuming $\sqrt{\delta_\tau} \geq 2\epsilon$ and using (3.6) and bounding the failure probability with a union bound, we get (w.p. $\geq 1 - \tau \cdot C/T$)

$$\delta_{\tau+1} \leq \frac{\delta_\tau(\sigma^2 + 1/2)^2}{(1-\delta_\tau)(\sigma^2+3/4)^2 + \delta_\tau(\sigma^2+1/2)^2} \overset{(i)}{\leq} \frac{\gamma^{2\tau}\delta_0}{1-(1-\gamma^{2\tau})\delta_0} \overset{(ii)}{\leq} C_1\gamma^{2\tau}p, \tag{3.7}$$

where $\gamma = \frac{\sigma^2+1/2}{\sigma^2+3/4}$ and $C_1 > 0$ is a global constant. Inequality (ii) follows from Lemma 5; to prove (i), we need the following lemma. It shows that in the recursion given by (3.7), δ_τ decreases at a fast rate. The rate of decrease in δ_τ might be initially (for small τ) sub-linear, but for large enough τ the rate is linear. We defer the proof to Section 3.4.

Lemma 6. *If for any $\tau \geq 0$ and $0 < \gamma < 1$, we have $\delta_{\tau+1} \leq \frac{\gamma^2\delta_\tau}{1-\delta_\tau+\gamma^2\delta_\tau}$, then,*

$$\delta_{\tau+1} \leq \frac{\gamma^{2t+2}\delta_0}{1-(1-\gamma^{2t+2})\delta_0}.$$

Hence, using the above equation after $T = O\left(\log(p/\epsilon)/\log\left(1/\gamma\right)\right)$ updates, with probability at least $1 - C$, $\sqrt{\delta_T} \leq 2\epsilon$. The result now follows by noting that $\|\mathbf{u} - \mathbf{q}_T\|_2 \leq 2\sqrt{\delta_T}$. □

Remark: In Theorem 2, the probability of recovery is a constant and does not decay with p. One can correct this by either paying a price of $O(\log p)$ in storage, or in sample complexity: for the former, we can run $O(\log p)$ instances of Algorithm 1 in parallel; alternatively, we can run Algorithm 1

$O(\log p)$ times on fresh data each time, using the next block of data to evaluate the old solutions, always keeping the best one. Either approach guarantees a success probability of at least $1 - \frac{1}{p^{O(1)}}$.

3.3.2 General Rank-k Case

In this section, we consider the general rank-k PCA problem where each sample is assumed to be generated using the model of equation (3.1), where $A \in \mathbb{R}^{p \times k}$ represents the k principal components that need to be recovered. Let $A = U\Lambda V^\top$ be the SVD of A where $U \in \mathbb{R}^{p \times k}$, $\Lambda, V \in \mathbb{R}^{k \times k}$. The matrices U and V are orthogonal, i.e., $U^\top U = I, V^\top V = I$, and Σ is a diagonal matrix with diagonal elements $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_k$. The goal is to recover the space spanned by A, i.e., span(U). Without loss of generality, we can assume that $\|A\|_2 = \lambda_1 = 1$.

Similar to the rank-1 problem, our algorithm for the rank-k problem can be viewed as a streaming variant of the classical orthogonal iteration used for SVD. But unlike the rank-1 case, we require a more careful analysis as we need to bound spectral norms of various quantities in intermediate steps and simple, crude analysis can lead to significantly worse bounds. Interestingly, the analysis is entirely different from the standard analysis of the orthogonal iteration as there, the empirical estimate of the covariance matrix is fixed while in our case it varies with each block.

For the general rank-k problem, we use the largest-principal-angle-

based distance function between any two given subspaces:

$$\text{dist}\left(\text{span}(U), \text{span}(V)\right) = \text{dist}(U, V) = \|U_\perp^\top V\|_2 = \|V_\perp^\top U\|_2,$$

where $U_\perp$ and $V_\perp$ represent an orthogonal basis of the perpendicular subspace to span(U) and span(V), respectively. For the spiked covariance model, it is straightforward to see that this is equivalent to the usual PCA figure-of-merit, the expressed variance.

Theorem 7. *Consider a data stream, where $\mathbf{x}_t \in \mathbb{R}^p$ for every t is generated by (3.1), and the SVD of $A \in \mathbb{R}^{p\times k}$ is given by $A = U\Lambda V^\top$. Let, wlog, $\lambda_1 = 1 \geq \lambda_2 \geq \cdots \geq \lambda_k > 0$. Let,*

$$T = \Omega\left(\log\left(\frac{p}{k\epsilon}\right) / \log\left(\frac{\sigma^2 + 0.75\lambda_k^2}{\sigma^2 + 0.5\lambda_k^2}\right)\right),$$

$$B = \Omega\left(\frac{\left((1+\sigma)^2\sqrt{k} + \sigma\sqrt{1+\sigma^2}k\sqrt{p}\right)^2 \log(T)}{\lambda_k^4\epsilon^2}\right).$$

Then, after T B-size-block-updates, w.p. 0.99, dist$(U, Q_T) \leq \epsilon$. Hence, the sufficient number of samples for ϵ-accurate recovery of all the top-k principal components is:

$$n = \tilde{\Omega}\left(\frac{\left((1+\sigma)^2\sqrt{k} + \sigma\sqrt{1+\sigma^2}k\sqrt{p}\right)^2 \log(p/k\epsilon)}{\lambda_k^4\epsilon^2 \log\left(\frac{\sigma^2+0.75\lambda_k^2}{\sigma^2+0.5\lambda_k^2}\right)}\right).$$

Again, we use $\tilde{\Omega}(\cdot)$ to suppress the extra $\log(T)$ factor.

The key part of the proof requires the following additional lemmas that bound the energy of the current iterate along the desired subspace and

its perpendicular space (Lemmas 8 and 9), and Lemma 10, which controls the quality of the initialization.

Lemma 8. *Let $\mathcal{X}$, A, B, and T be as defined in Theorem 7. Also, let σ be the variance of noise, $F_{\tau+1} = \frac{1}{B}\sum_{B\tau<t\leq B(\tau+1)} \mathbf{x}_t\mathbf{x}_t^\top$ and Q_τ be the τ-th iterate of Algorithm 1. Then, $\forall\ \mathbf{v} \in \mathbb{R}^k$ and $\|\mathbf{v}\|_2 = 1$, w.p. $1 - 5C/T$ we have:*

$$\|U^\top F_{\tau+1} Q_\tau \mathbf{v}\|_2 \geq (\lambda_k^2 + \sigma^2 - \frac{\lambda_k^2\epsilon}{4})\sqrt{1 - \|U_\perp^\top Q_\tau\|_2^2}.$$

Lemma 9. *Let $\mathcal{X}$, A, B, $F_{\tau+1}$, Q_τ be as defined in Lemma 8. Then, w.p. $1 - 4C/T$, $\|U_\perp^\top F_{\tau+1} Q_\tau\|_2 \leq \sigma^2\|U_\perp^\top Q_\tau\|_2 + \lambda_k^2\epsilon/2$.*

Lemma 10. *Let $Q_0 \in \mathbb{R}^{p\times k}$ be sampled uniformly at random from the set of all k-dimensional subspaces (see Initialization Steps of Algorithm 1). Then, w.p. at least 0.99: $\sigma_k(U^\top Q_0) \geq C\sqrt{\frac{1}{kp}}$, where $C > 0$ is a global constant.*

The full proofs for all the lemmata and Theorem 7 are given in Section 3.4.

3.4 Proofs

In this section we provide all so far omitted proofs. First, we provide some results from the literature – what we call Preliminaries – then we prove Theorem 7 and supporting lemmas.

3.4.1 Preliminaries

Lemma 11 (Lemma 5.4 of [73]). *Let A be a symmetric $k \times k$ matrix, and let $\mathcal{N}_\epsilon$ be an ϵ-net of S^{k-1} for some $\epsilon \in [0,1)$. Then,*

$$\|A\|_2 \leq \frac{1}{(1-2\epsilon)} \sup_{\mathbf{x} \in \mathcal{N}_\epsilon} |\langle A\mathbf{x}, \mathbf{x} \rangle|.$$

Lemma 12 (Proposition 2.1 of [72]). *Consider independent random vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $\mathbb{R}^p$, $n \geq p$, which have sub-Gaussian distribution with parameter 1. Then for every $\delta > 0$ with probability at least $1 - \delta$ one has,*

$$\|\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T - \mathbb{E}[\mathbf{x}_i \mathbf{x}_i^T]\|_2 \leq C \sqrt{\log(2/\delta)} \sqrt{\frac{p}{n}}.$$

Lemma 13 (Corollary 3.5 of [73]). *Let A be an $N \times n$ matrix whose entries are independent standard normal random variables. Then for every $t \geq 0$, with probability at least $1 - 2\exp(-t^2/2)$ one has,*

$$\sqrt{N} - \sqrt{n} - t \leq \sigma_k(A) \leq \sigma_1(A) \leq \sqrt{N} + \sqrt{n} + t.$$

Lemma 14 (Theorem 1.2 of [63]). *Let $\zeta_1, \ldots, \zeta_n$ be independent centered real random variables with variances at least 1 and subgaussian moments bounded by B. Let A be an $k \times k$ matrix whose rows are independent copies of the random vector $(\zeta_1, \ldots, \zeta_n)$. Then for every $\epsilon \geq 0$ one has*

$$\Pr(\sigma_{min}(A) \leq \epsilon/\sqrt{k}) \leq C\epsilon + c^n,$$

where $C > 0$ and $c \in (0,1)$ depend only on B. Note that $B = 1$ for the standard Gaussian variables.

Lemma 15. *Let $\mathbf{x}_i \in \mathbb{R}^m, 1 \leq i \leq B$ be i.i.d. standard multivariate normal variables. Also, $\mathbf{y}_i \in \mathbb{R}^n$ are also i.i.d. normal variables and are independent of $\mathbf{x}_i, \forall i$. Then, w.p. $1-\delta$,*

$$\left\| \frac{1}{B} \sum_i \mathbf{x}_i \mathbf{y}_i^\top \right\|_2 \leq \sqrt{\frac{C \max(m,n) \log(2/\delta)}{B}}.$$

Proof. Let $M = \sum_i \mathbf{x}_i \mathbf{y}_i^T$ and let $m > n$. Then, the goal is to show that, the following holds w.p. $1-\delta$: $\frac{1}{B}\|M\mathbf{v}\|_2 \leq \sqrt{\frac{Cm\log(2/\delta)}{B}}$ for all $\mathbf{v} \in \mathbb{R}^n$ s.t. $\|\mathbf{v}\|_2 = 1$.

We prove the lemma by first showing that the above mentioned result holds for any *fixed* vector v and then use standard epsilon-net argument to prove it for all $\mathbf{v}$.

Let $\mathcal{N}$ be the 1/4-net of S^{n-1}. Then, using Lemma 5.4 of [73] (see Lemma 11),

$$\|\frac{1}{Bm} M^T M\|_2 \leq 2 \max_{\mathbf{v} \in \mathcal{N}} \frac{1}{Bm} \|M\mathbf{v}\|_2^2. \tag{3.8}$$

Now, for any fixed $\mathbf{v}$: $M\mathbf{v} = \sum_i \mathbf{x}_i \mathbf{y}_i^T \mathbf{v} = \sum_i \mathbf{x}_i c_i$, where $c_i = \mathbf{y}_i^T \mathbf{v} \sim N(0,1)$. Hence,

$$\|M\mathbf{v}\|_2^2 = \sum_{\ell=1}^{m} (\sum_{i=1}^{B} x_{i\ell} c_i)^2.$$

Now, $\sum_{i=1}^{B} x_{i\ell} c_i \sim N(0, \|c\|_2^2)$ where $c^T = [c_1\ c_2 \cdots c_B]$. Hence, $\sum_{i=1}^{B} x_{i\ell} c_i = \|c\|_2 h_\ell$ where $h_\ell \sim N(0,1)$.

Therefore, $\|M\mathbf{v}\|_2^2 = \|c\|_2^2\|h\|_2^2$ where $h^T = [h_1\ h_2 \cdots h_B]$. Now,

$$Pr(\frac{\|c\|_2^2\|h\|_2^2}{Bm} \geq 1+\gamma) \leq Pr(\frac{\|c\|_2^2}{B} \geq \sqrt{1+\gamma}) + Pr(\frac{\|h\|_2^2}{m} \geq \sqrt{1+\gamma}) \\ \overset{\zeta_1}{\leq} 2\exp(-\frac{B\gamma^2}{32}) + 2\exp(-\frac{m\gamma^2}{32}) \leq 4\exp(-\frac{m\gamma^2}{32}), \quad (3.9)$$

where $0 < \gamma < 3$ and ζ_1 follows from Lemma 13.

Using (3.8), (3.9), the following holds with probability $(1 - 9^{n+1}e^{-\frac{m\gamma^2}{32}})$:

$$\frac{\|M\|_2^2}{Bm} \leq 1 + 2\gamma. \quad (3.10)$$

The result now follows by setting γ appropriately and assuming $n < Cm$ for small enough C. □

3.4.2 Proof of Theorem 7

Recall that our algorithm proceeds in a blockwise manner; for each block of samples, we compute

$$S_{\tau+1} = \left(\frac{1}{B}\sum_{t=B\tau+1}^{B(\tau+1)} \mathbf{x}_t\mathbf{x}_t^\top\right) Q_\tau, \quad (3.11)$$

where $Q_\tau \in \mathbb{R}^{p\times k}$ is the τ-th block iterate and is an orthogonal matrix, i.e., $Q_\tau^\top Q_\tau = I_{k\times k}$. Given $S_{\tau+1}$, the next iterate, $Q_{\tau+1}$, is computed by the QR-decomposition of $S_{\tau+1}$. That is,

$$S_{\tau+1} = Q_{\tau+1}R_{\tau+1}, \quad (3.12)$$

where $R_{\tau+1} \in \mathbb{R}^{k\times k}$ is an upper-triangular matrix.

Proof. By using update for $Q_{\tau+1}$ (see (3.11), (3.12)):

$$Q_{\tau+1}R_{\tau+1} = F_{\tau+1}Q_\tau, \tag{3.13}$$

where $F_{\tau+1} = \frac{1}{B}\sum_{B\tau<t\leq B(\tau+1)} \mathbf{x}_t\mathbf{x}_t^\top$. That is,

$$U_\perp^\top Q_{\tau+1}R_{\tau+1}\mathbf{v} = U_\perp^\top F_{\tau+1}Q_\tau\mathbf{v}, \quad \forall \mathbf{v}\in\mathbb{R}^k, \tag{3.14}$$

where $U_\perp$ is an orthogonal basis of the subspace orthogonal to span(U). Now, let $\mathbf{v}_1$ be the singular vector corresponding to the largest singular value, then:

$$\begin{aligned}
\|U_\perp^\top Q_{\tau+1}\|_2^2 &= \frac{\|U_\perp^\top Q_{\tau+1}\mathbf{v}_1\|_2^2}{\|\mathbf{v}_1\|_2^2} = \frac{\|U_\perp^\top Q_{\tau+1}R_{\tau+1}\tilde{\mathbf{v}}_1\|_2^2}{\|R_{\tau+1}\tilde{\mathbf{v}}_1\|_2^2} \\
&\overset{(i)}{=} \frac{\|U_\perp^\top Q_{\tau+1}R_{\tau+1}\tilde{\mathbf{v}}_1\|_2^2}{\|U^\top Q_{\tau+1}R_{\tau+1}\tilde{\mathbf{v}}_1\|_2^2 + \|U_\perp^\top Q_{\tau+1}R_{\tau+1}\tilde{\mathbf{v}}_1\|_2^2} \\
&\overset{(ii)}{=} \frac{\|U_\perp^\top F_{\tau+1}Q_\tau\tilde{\mathbf{v}}_1\|_2^2}{\|U^\top F_{\tau+1}Q_\tau\tilde{\mathbf{v}}_1\|_2^2 + \|U_\perp^\top F_{\tau+1}Q_\tau\tilde{\mathbf{v}}_1\|_2^2}.
\end{aligned} \tag{3.15}$$

where $\tilde{\mathbf{v}}_1 = \frac{R_{\tau+1}^{-1}\mathbf{v}_1}{\|R_{\tau+1}^{-1}\mathbf{v}_1\|_2}$. (i) follows as $Q_{\tau+1}$ is an orthogonal matrix and $[U\ U_\perp]$ form a complete orthogonal basis; (ii) follows by using (3.13). The existence of $R_{\tau+1}^{-1}$ follows using Lemma 8 along with the fact that $\sigma_k(R_{\tau+1}) = \|R_{\tau+1}\zeta_0\|_2 \geq \|U^\top Q_{\tau+1}R_{\tau+1}\zeta_0\|_2 = \|U^\top F_{\tau+1}Q_\tau\zeta_0\|_2 > 0$, where ζ_0 is the singular vector of $R_{\tau+1}$ corresponding to its smallest singular value, $\sigma_k(R_{\tau+1})$.

Now, using (3.15) with Lemmas 8, 9 and using the fact that $x/(x+c)$ is an increasing function of x, for all $x > 0$, we get (w.p. $\geq 1 - 2C/T$):

$$\|U_\perp^\top Q_{\tau+1}\|_2^2 \leq \frac{(\sigma^2\|U_\perp^\top Q_\tau\|_2 + \lambda_k^2\epsilon/2)^2}{(\lambda_k^2 + \sigma^2 - \frac{\lambda_k^2\epsilon}{4})^2(1 - \|U_\perp^\top Q_\tau\|_2^2) + (\sigma^2\|U_\perp^\top Q_\tau\|_2 + 0.5\lambda_k^2\epsilon)^2}.$$

Now, assuming $\epsilon \leq \|U_\perp^\top Q_\tau\|_2^2$, using the above equation and by using union bound, we get (w.p. $\geq 1 - 2\tau C/T$):

$$\|U_\perp^\top Q_{\tau+1}\|_2^2 \leq \frac{\gamma^2\|U_\perp^\top Q_\tau\|_2^2}{1 - \|U_\perp^\top Q_\tau\|_2^2 + \gamma^2\|U_\perp^\top Q_\tau\|_2^2}, \tag{3.16}$$

where $\gamma = \frac{\sigma^2+\lambda_k^2/2}{\sigma^2+3\lambda_k^2/4} < 1$ for $\lambda_k > 0$. Using Lemma 6 along with the above equation, we get (w.p. $\geq 1 - 2\tau C/T$):

$$\|U_\perp^\top Q_{\tau+1}\|_2^2 \leq \gamma^{2\tau} \frac{\|U_\perp^\top Q_0\|_2^2}{1 - \|U_\perp^\top Q_0\|_2^2}.$$

Now, using Lemma 10 we know that $\|U_\perp^\top Q_0\|_2^2$ is at most $1-\Omega(1/(kp))$. Hence, for $T = O(\log(p/\epsilon)/\log(1/\gamma)$, we get: $\|U_\perp^\top Q_T\|_2^2 \leq \epsilon$. Furthermore, we require B (as mentioned in the Theorem) samples per block. Hence, the total sample complexity bound is given by $O(BT)$, concluding the proof. □

3.4.3 Proof of Lemma 3

Proof. Note that,

$$\frac{1}{B}\sum_t \mathbf{x}_t\mathbf{x}_t^\top - \mathbf{u}\mathbf{u}^\top - \sigma^2 I = \mathbf{u}\mathbf{u}^\top \frac{1}{B}\sum_t (z_t^2 - 1) + \frac{1}{B}\sum_t (\mathbf{w}_t\mathbf{w}_t^\top - \sigma^2 I) + \frac{1}{B}\sum_t z_t \mathbf{w}_t\mathbf{u}^\top + \frac{1}{B}\mathbf{u}\sum_t z_t \mathbf{w}_t^\top. \quad (3.17)$$

We now individually bound each of the above given terms in the RHS. Using standard tail bounds for covariance estimation (see Lemma 12), we can bound the first two terms (w.p. $1 - 2C/T$):

$$\frac{1}{B}\left|\sum_t (z_t^2 - 1)\right| \leq \sqrt{\frac{C\log(T)}{B}},$$
$$\|\frac{1}{B}\sum_t (\mathbf{w}_t\mathbf{w}_t^\top - \sigma^2 I)\|_2 \leq \sigma^2\sqrt{\frac{C_1 p\log(T)}{B}}. \quad (3.18)$$

Similarly, using Lemma 15, we can bound the last two terms in (3.17) (w.p. $1 - 2C/T$):

$$\|\frac{1}{B}\sum_t z_t\mathbf{w}_t\mathbf{u}^\top\|_2 = \|\frac{1}{B}\mathbf{u}\sum_t z_t\mathbf{w}_t^\top\|_2 \leq \sigma\sqrt{\frac{C_1 p\log(T)}{B}}. \quad (3.19)$$

The lemma now follows by using (3.17), (3.18), (3.19) along with B as given by Theorem 2. □

3.4.4 Proof of Lemma 4

Proof. Let $\mathbf{q}_\tau = \sqrt{1-\delta_\tau}\mathbf{u} + \sqrt{\delta_\tau}\mathbf{u}_\tau^\perp$, where $\mathbf{u}_\tau^\perp$ is the component of $\mathbf{q}_\tau$ that is orthogonal to $\mathbf{u}$. Now,

$$\begin{aligned}
\mathbf{u}^\top \mathbf{s}_{\tau+1} &= \frac{1}{B}\sum_t (\mathbf{u}^\top \mathbf{x}_t)(\mathbf{x}_t^\top \mathbf{q}_t) \\
&= \frac{1}{B}\sum_t (z_t + \mathbf{u}^\top \mathbf{w}_t)(\sqrt{1-\delta_\tau}(z_t + \mathbf{u}^\top \mathbf{w}_t) + \sqrt{\delta_\tau}\mathbf{w}_t^\top \mathbf{u}_\tau^\perp) \\
&= \frac{\sqrt{1-\delta_\tau}}{B}\sum_t (z_t + \mathbf{u}^\top \mathbf{w}_t)^2 + \frac{\sqrt{\delta_\tau}}{B}\sum_t (z_t + \mathbf{u}^\top \mathbf{w}_t)\mathbf{w}_t^\top \mathbf{u}_\tau^\perp. \qquad (3.20)
\end{aligned}$$

Now, the first term above is a summation of B i.i.d. chi-square variables and hence using standard results (see Lemma 13), w.p. $(1 - C/T)$:

$$\frac{1}{B}\sum_t (z_t + \mathbf{u}^\top \mathbf{w}_t)^2 \geq (1+\sigma^2)(1 - \sqrt{\frac{C\log(2T)}{B}}). \qquad (3.21)$$

Also, $\mathbf{w}_t^\top \mathbf{u}$ and $\mathbf{w}_t^\top \mathbf{u}_\tau^\perp$ are independent random variables, as both $\mathbf{w}_t^\top \mathbf{u}$, $\mathbf{w}_t^\top \mathbf{u}_\tau^\perp$ are Gaussians and $E[\mathbf{w}_t^\top \mathbf{u}_\tau^\perp \mathbf{u}^\top \mathbf{w}_t] = 0$. Hence, using Lemma 15, the following holds with probability $\geq 1 - 4C/T$:

$$\begin{aligned}
\|\frac{1}{B}\sum_t (z_t + \mathbf{u}^\top \mathbf{w}_t)\mathbf{w}_t^\top \mathbf{u}_\tau^\perp\|_2 &\leq \sigma\sqrt{1+\sigma^2}\sqrt{\frac{C\log(T)}{B}} \\
&\overset{(i)}{\leq} \sigma\sqrt{1+\sigma^2}\sqrt{\frac{C_1 p\log(T)}{B(1-\delta_0)}}\sqrt{1-\delta_\tau}, \qquad (3.22)
\end{aligned}$$

where (i) follows by using inductive hypothesis (i.e., $\sqrt{1-\delta_\tau} > \sqrt{1-\delta_{\tau-1}}$, induction step follows as we show that the error decreases at each step) and Lemma 5.

The lemma now follows by using (3.20), (3.21), (3.22) and by setting B, T appropriately. □

3.4.5 Proof of Lemma 5

Proof. Using standard tail bounds for Gaussians (see Lemma 13), $\|\mathbf{q}_0\|_2 \leq 2\sqrt{p}$ with probability $1 - \exp(-C_1 p)$, where $C_1 > 0$ is a universal constant. Furthermore, $(\|\mathbf{q}_0\|_2 \mathbf{q}_0)^\top \mathbf{u} \sim N(0,1)$. Hence, there exists $C_0 > 0$, s.t., with probability 0.99, $|(\|\mathbf{q}_0\|_2 \mathbf{q}_0)^T \mathbf{u}| \geq C_0$. Hence, $|\mathbf{q}_0^\top \mathbf{u}| \geq \frac{C_0}{2\sqrt{p}}$. □

3.4.6 Proof of Lemma 6

Proof. We prove the lemma using induction. The base case (for $\tau = 0$) follows trivially.

Now, by the inductive hypothesis, $\delta_\tau \leq \frac{\gamma^{2t}\delta_0}{1-(1-\gamma^{2t})\delta_0}$. That is,

$$\frac{1}{\delta_\tau} \geq \frac{1-(1-\gamma^{2t})\delta_0}{\gamma^{2t}\delta_0}.$$

Finally, by assumption,

$$\delta_{\tau+1} \leq \frac{\gamma^2}{\frac{1}{\delta_\tau} - (1-\gamma^2)} \leq \frac{\gamma^2}{\frac{1-(1-\gamma^{2t})\delta_0}{\gamma^{2t}\delta_0} - (1-\gamma^2)}.$$

The lemma follows after simplification of the above given expression. □

3.4.7 Proof of Lemma 8

Proof. Using the generative model (3.1), we get:

$$U^\top F_{\tau+1} Q_\tau \mathbf{v} = \Lambda \left(\frac{1}{B} \sum_t \mathbf{z}_t \mathbf{z}_t^\top \right) \Lambda U^\top Q_\tau \mathbf{v} + \left(\frac{1}{B} \sum_t U^\top \mathbf{w}_t \mathbf{w}_t^\top U \right) U^\top Q_\tau \mathbf{v} + (\frac{1}{B} \sum_t U^\top \mathbf{w}_t \mathbf{z}_t^\top) \Lambda U^\top Q_\tau \mathbf{v} + \Lambda (\frac{1}{B} \sum_t \mathbf{z}_t \mathbf{w}_t^\top U) U^\top Q_\tau \mathbf{v} + \left(\frac{1}{B} \sum_t (\Lambda \mathbf{z}_t + U^\top \mathbf{w}_t) \mathbf{w}_t^\top U_\perp U_\perp^\top Q_\tau \right) \mathbf{v}. \quad (3.23)$$

Note that in the equation and rest of the proof, t varies from $B\tau < t \leq B(\tau+1)$.

We now show that each of the five terms in the above given equation concentrate around their respective means. Also, let $\mathbf{y}_t = U^\top \mathbf{w}_t$ and $\mathbf{y}_t^\perp = U_\perp^\top \mathbf{w}_t$. Note that, $\mathbf{y}_t \sim N(0, \sigma^2 I_{k\times k})$ and $\mathbf{y}_t^\perp \sim N(0, \sigma^2 I_{(p-k)\times(p-k)})$.

(a): Consider the first term in (3.23). Using $\|A\mathbf{v}\|_2 \leq \|A\|_2 \|\mathbf{v}\|_2$ and the assumption that $\lambda_1 = 1$, we get:

$$\|\Lambda \left(\frac{1}{B} \sum_t \mathbf{z}_t \mathbf{z}_t^\top - I \right) \Lambda U^\top Q_\tau \mathbf{v}\|_2 \leq \| \left(\frac{1}{B} \sum_t \mathbf{z}_t \mathbf{z}_t^\top - I \right) \|_2 \|U^\top Q_\tau \mathbf{v}\|_2$$

. Using Lemma 12 we get (w.p. $1 - C/T$):

$$\| \frac{1}{B} \sum_t \mathbf{z}_t \mathbf{z}_t^\top - I \|_2 \leq \sqrt{\frac{C_1 k \log(T)}{B}}.$$

That is,

$$\|\Lambda (\frac{1}{B} \sum_t \mathbf{z}_t \mathbf{z}_t^\top - I) \Lambda U^\top Q_\tau \mathbf{v}\|_2 \leq \sqrt{\frac{C_1 k \log(T)}{B}} \|U^\top Q_\tau \mathbf{v}\|_2. \quad (3.24)$$

(b): Similarly, the second term in (3.23) can be bounded as (w.p. $1 - C/T$):

$$\| \left(\frac{1}{B} \sum_t U^\top \mathbf{w}_t \mathbf{w}_t^\top U - \sigma^2 I \right) U^\top Q_\tau \mathbf{v}\|_2 \leq \sigma^2 \sqrt{\frac{C_1 k \log(T)}{B}} \|U^\top Q_\tau \mathbf{v}\|_2. \quad (3.25)$$

(c): Now consider the third and the fourth term. Now $\mathbf{w}_t$ and $\mathbf{z}_t$ are independent 0-mean Gaussians, hence using Lemma 15, we get: $\|\frac{1}{B}\sum_t U^\top \mathbf{w}_t \mathbf{z}_t^\top\|_2 \leq \sigma\sqrt{\frac{C_1 k\log(T)}{B}}$. Hence, w.p. $1-2C/T$,

$$\|\Lambda(\frac{1}{B}\sum_t \mathbf{z}_t\mathbf{w}_t^\top U)U^\top Q_\tau \mathbf{v}\| + \|(\frac{1}{B}\sum_t U^\top \mathbf{w}_t\mathbf{z}_t)\Lambda U^\top Q_\tau \mathbf{v}\| \leq 2\sigma\sqrt{\frac{C_1 k\log(T)}{B}}\|U^\top Q_\tau \mathbf{v}\|_2. \tag{3.26}$$

(d): Finally, we consider the last term in (3.23). Note that, $(\Lambda\mathbf{z}_t + U^\top\mathbf{w}_t) \sim N(0, D)$ where D is a diagonal matrix with $D_{ii} = \lambda_i^2+\sigma^2$. Also, $Q^\top U_\perp U_\perp^\top \mathbf{w}_t \sim N(0, \sigma^2 I_{(p-k)\times(p-k)})$ and is independent of $(\Lambda\mathbf{z}_t + U^\top\mathbf{w}_t)$ as $E[Q^\top U_\perp U_\perp^\top \mathbf{w}_t\mathbf{w}_t^\top U] = 0$; recall that for Gaussian RVs, covariance is zero iff RVs are independent. Hence, using Lemma 15, w.p. $\geq 1-C/T$:

$$\|(\frac{1}{B}\sum_t(\Lambda\mathbf{z}_t + U^\top\mathbf{w}_t)\mathbf{w}_t^\top U_\perp U_\perp^\top Q_\tau)\mathbf{v}\|_2 \leq \sqrt{1+\sigma^2}\sigma\sqrt{\frac{C_1 k\log(T)}{B}}. \tag{3.27}$$

Now, using (3.23), (3.24), (3.25), (3.26), (3.27) (w.p. $\geq 1-5C/T$)

$$\|U^\top F_{\tau+1}Q_\tau\mathbf{v}\|_2 \geq \|(\Lambda^2+\sigma^2 I)U^\top Q_\tau\mathbf{v}\|_2 - \sqrt{\frac{C_1 k\log(T)}{B}}\|U^\top Q_\tau\mathbf{v}\|_2\left((1+\sigma)^2 + \frac{\sigma\sqrt{1+\sigma^2}}{\|U^\top Q_\tau\mathbf{v}\|_2}\right). \tag{3.28}$$

Now, $\|U^\top Q_\tau\mathbf{v}\|_2 \geq \sigma_k(U^\top Q_\tau\mathbf{v})$. Next, by using the inductive hypothesis (i.e., $\sigma_k(U^T Q_\tau) \geq \sigma_k(U^T Q_{\tau-1})$, induction step follows as we show that the error decreases at each step) and Lemma 10, we have $\|U^\top Q_\tau\mathbf{v}\|_2 \geq \sigma_k(U^\top Q_0) \geq \frac{C}{\sqrt{pk}}$ with probability ≥ 0.99.

Also, $\|(\Lambda^2+\sigma^2 I)U^\top Q_\tau\mathbf{v}\|_2 \geq (\lambda_k^2+\sigma^2)\|U^\top Q_\tau\mathbf{v}\|_2$. Additionally, $\|U^\top Q_\tau\mathbf{v}\|_2 \geq \sqrt{1-\|U_\perp^\top Q_\tau\|_2^2}$. Hence, lemma follows by using these facts with (3.28) and by selecting B as given in Theorem 7. □

3.4.8 Proof of Lemma 9

Proof. Similar to our proof for Lemma 8, we separate out the "error" or deviation terms in $\|U_\perp^\top F_{\tau+1} Q_\tau\|_2$ and bound them using concentration bounds. Now,

$$\begin{aligned}\|U_\perp^\top F_{\tau+1} Q_\tau \mathbf{v}\|_2 &= \|U_\perp^\top (U\Lambda^2 U^\top + \sigma^2 I + E_\tau) Q_\tau \mathbf{v}\|_2 \\ &\leq \|\sigma^2 U_\perp^\top Q_\tau \mathbf{v}\|_2 + \|U_\perp^\top E_\tau Q_\tau \mathbf{v}\|_2 \\ &\leq \sigma^2 \|U_\perp^\top Q_\tau \mathbf{v}\|_2 + \|E_\tau\|_2, \end{aligned} \tag{3.29}$$

where E_τ is the error matrix representing deviation of the estimate $F_{\tau+1}$ from its mean. That is,

$$\begin{aligned} E &= \frac{1}{B}\sum_t \mathbf{x}_t \mathbf{x}_t^\top - U\Lambda^2 U^\top - \sigma^2 I \\ &= U\Lambda(\frac{1}{B}\sum_t \mathbf{z}_t \mathbf{z}_t^\top - I)\Lambda U^\top + (\frac{1}{B}\sum_t \mathbf{w}_t \mathbf{w}_t^\top - \sigma^2 I) \\ &+ U\Lambda \frac{1}{B}\sum_t \mathbf{z}_t \mathbf{w}_t^\top + \frac{1}{B}\sum_t \mathbf{w}_t \mathbf{z}_t^\top \Lambda U. \end{aligned} \tag{3.30}$$

Note that the above given four terms correspond to similar four terms in (3.23) and hence can be bounded in similar fashion. In particular, the following holds with probability $1 - 4C/T$:

$$\|E\|_2 \leq \sqrt{\frac{C_1 k \log(T)}{B}} + \sigma^2 \sqrt{\frac{C_1 p \log(T)}{B}} + 2\sigma \sqrt{\frac{C_1 p \log(T)}{B}} \leq \lambda_k^2 \epsilon/2, \tag{3.31}$$

where the second inequality follows by setting B as required by Theorem 7. The lemma now follows using (3.29), (3.30), (3.31). □

3.4.9 Proof of Lemma 10

Proof. Using Step 2 of Algorithm 1: $H = Q_0 R_0$. Let $\mathbf{v}_k$ be the singular vector of $U^\top Q_0$ corresponding to the smallest singular value. Then,

$$\begin{aligned}\sigma_k(U^\top Q_0) &= \frac{\|U^\top Q_0 R_0 R_0^{-1}\mathbf{v}_k\|_2}{\|R_0^{-1}\mathbf{v}_k\|_2}\|R_0^{-1}\mathbf{v}_k\|_2 \\ &\geq \sigma_k(U^\top Q_0 R_0)\sigma_k(R_0^{-1}). \end{aligned} \tag{3.32}$$

Now, $\sigma_k(R_0^{-1}) = \frac{1}{\|R_0\|_2} = \frac{1}{\|Q_0 R_0\|_2} = \frac{1}{\|H\|_2}$. Note that $\|H\|_2$ is the spectral norm of a random matrix with i.i.d. Gaussian entries and hence can be easily bounded using standard results. In particular, using Lemma 13, we get: $\|H\|_2 \leq C_1\sqrt{p}$ w.p. $\geq 1 - e^{-C_2 p}$, where $C_1, C_2 > 0$ are global constants.

By Theorem 1.1 of [63] (see Lemma 14), w.p. ≥ 0.99, $\sigma_k(U^\top Q_0 R_0) = \sigma_k(H) \geq C/\sqrt{k}$. The lemma now follows using the above two bounds with (3.32). □

3.5 Perturbation-tolerant Subspace Recovery

While our results thus far assume A has rank exactly k, and k is known *a priori*, here we show that both these can be relaxed; hence our results hold in a quite broad setting.

Let $\mathbf{x}_t = A\mathbf{z}_t + \mathbf{w}_t$ be the t-th step sample, with $A = U\Lambda V^T \in \mathbb{R}^{p\times r}$ and $U \in \mathbb{R}^{p\times r}$, where $r \geq k$ is the unknown true rank of A. We run Algorithm 1 with rank k to recover a subspace Q_T that is contained in U. The largest principal angle-based distance, from the previous section, can be used directly

in our more general setting. That is, $\text{dist}(U, Q_T) = \|U_\perp^T Q_T\|_2$ measures the component of Q_T "outside" the subspace U.

Now, our analysis can be easily modified to handle this case. Naturally, now the number of samples we require increases according to r. In particular, if

$$n = \tilde{\Omega}\left(\frac{\left((1+\sigma)^2\sqrt{r} + \sigma\sqrt{1+\sigma^2}r\sqrt{p}\right)^2 \log(p/r\epsilon)}{\lambda_r^4 \epsilon^2 \log\left(\frac{\sigma^2+0.75\lambda_r^2}{\sigma^2+0.5\lambda_r^2}\right)}\right),$$

then $\text{dist}(U, Q_T) \leq \epsilon$. Furthermore, if we assume $r \geq C \cdot k$ (or a large enough constant $C > 0$) then the initialization step provides us better distance, i.e., $\text{dist}(U, Q_0) \leq C'/\sqrt{p}$ rather than $\text{dist}(U, Q_0) \leq C'/\sqrt{kp}$ bound if $r = k$. This initialization step enables us to give tighter sample complexity as the $r\sqrt{p}$ in the numerator above can be replaced by $\sqrt{rp}$.

3.6 Experiments

In this section, we show that, as predicted by our theoretical results, our algorithm performs close to the optimal batch SVD. We provide the results from simulating the spiked covariance model, and demonstrate the phase-transition in the probability of successful recovery that is inherent to the statistical problem. Then we stray from the analyzed model and performance metric and test our algorithm on real world–and some very big–datasets, using the metric of explained variance.

In the experiments for Figures 3.1 (a)-(b), we draw data from the generative model of (3.1). Our results are averaged over at least 200 independent

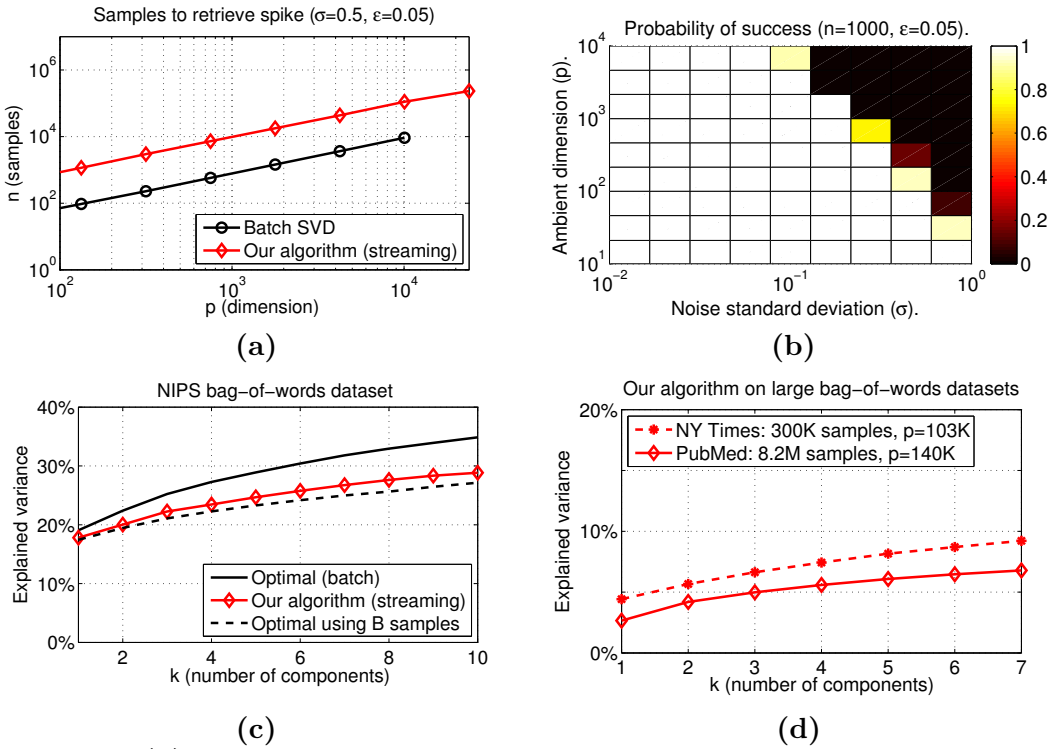


Figure 3.1: **(a)** Number of samples required for recovery of a single component ($k = 1$) from the spiked covariance model, with noise standard deviation $\sigma = 0.5$ and desired accuracy $\epsilon = 0.05$. **(b)** Fraction of trials in which Algorithm 1 successfully recovers the principal component ($k = 1$) in the same model, with $\epsilon = 0.05$ and $n = 1000$ samples, **(c)** Explained variance by Algorithm 1 compared to the optimal batch SVD, on the NIPS bag-of-words dataset. **(d)** Explained variance by Algorithm 1 on the NY Times and PubMed datasets.

runs. Algorithm 1 uses the block size prescribed in Theorem 7, with the empirically tuned constant of 0.2. As expected, our algorithm exhibits linear scaling with respect to the ambient dimension p – the same as the batch SVD. The missing point on batch SVD's curve (Figure 3.1(a)), corresponds to $p > 2.4 \cdot 10^4$. Performing SVD on a dense $p \times p$ matrix, either fails or takes a very long time on most modern desktop computers; in contrast, our

streaming algorithm easily runs on this size problem. The phase transition plot in Figure 3.1(b) shows the empirical sample complexity on a large class of problems and corroborates the scaling with respect to the noise variance we obtain theoretically.

Figures 3.1 (c)-(d) complement our complete treatment of the spiked covariance model, with some out-of-model experiments. We used three bag-of-words datasets from [58]. We evaluated our algorithm's performance with respect to the fraction of explained variance metric: given the $p \times k$ matrix V output from the algorithm, and all the provided samples in matrix X, the fraction of explained variance is defined as $\mathrm{Tr}(V^T X X^T V)/\mathrm{Tr}(X X^T)$. To be consistent with our theory, for a dataset of n samples of dimension p, we set the number of blocks to be $T = \lceil \log(p) \rceil$ and the size of blocks to $B = \lfloor n/T \rfloor$ in our algorithm. The NIPS dataset is the smallest, with 1500 documents and 12K words and allowed us to compare our algorithm with the optimal, batch SVD. We had the two algorithms work on the document space ($p = 1500$) and report the results in Figure 3.1(c). The dashed line represents the optimal using B samples. The figure is consistent with our theoretical result: our algorithm performs as well as the batch, with an added $\log(p)$ factor in the sample complexity.

Finally, in Figure 3.1 (d), we show our algorithm's ability to tackle very large problems. Both the NY Times and PubMed datasets are of prohibitive size for traditional batch methods – the latter including 8.2 million documents on a vocabulary of 141 thousand words – so we just report the performance

of Algorithm 1. It was able to extract the top 7 components for each dataset in a few hours on a desktop computer. A second pass was made on the data to evaluate the results, and we saw 7-10 percent of the variance explained on spaces with $p > 10^4$.

Chapter 4

Dealing with Highly Incomplete Samples

In this chapter we address a completely new set of challenges introduced by a stream of highly erased samples. Some algorithmic issues that arise are discussed in Section 2.2. In short, some traditional methods suffer from bias at the covariance estimation step. More importantly, methods that rely on imputing the missing entries, are bound to fail when faced with too many erasures (e.g. [12]). Here, we overcome of the above challenges.

We first formally pose the system model and objective, then present an algorithm for this problem and give theoretical guarantees for its convergence in Section 4.3. Finally, we present an extensive set of experiments that showcase our approach's many merits over the state of the art (Section 4.4).

4.1 Problem Formulation

System Model. Assume that at each time step t, we receive a point $\dot{\mathbf{x}}_t$, which is a partially erased version of $\mathbf{x}_t \in \mathbb{R}^p$. Our goal is to compute the top k principal components of the data: the k-dimensional subspace that offers the best squared-error estimate for the points. Our total storage capacity is $O(kp)$ – the storage required to store the output. The streaming setting means,

in particular, that any vector not explicitly stored can never be revisited.

Our analytical (sample complexity) guarantees are based on the following generative model for the data: the full samples are described by

$$\mathbf{x}_t = U\Lambda\mathbf{z}_t + \mathbf{w}_t, \tag{4.1}$$

where each component of $\mathbf{z}_t$, i.e., $\mathbf{z}_t^i, 1 \leq i \leq p$ is sampled i.i.d. from a fixed distribution $\mathcal{D}$, s.t., $\mathbb{E}[\mathbf{z}_t^i = 0]$, $\mathbb{E}[(\mathbf{z}_t^i)^2 = 1]$, and finally $|\mathbf{z}_t^i| \leq M_\infty$ almost surely. Similarly, we assume that each component of $\mathbf{w}_t$ is sampled i.i.d. from another fix distribution $\mathcal{D}'$ which also satisfies the same set of normalization constraints, i.e., $\mathbb{E}[\mathbf{w}_t^i = 0]$, $\mathbb{E}[(\mathbf{w}_t^i)^2 = 1]$, and $|\mathbf{w}_t^i| \leq M_\infty$ almost surely. The sequences $\{\mathbf{z}_t\}_t$ and $\{\mathbf{w}_t\}_t$ are mutually independent, $U \in \mathbb{R}^{p\times k}$ is a matrix with orthonormal columns and $\Lambda \in \mathbb{R}^{k\times k}$ a diagonal matrix.

Note that, our analysis holds even when $\mathbf{z}_t, \mathbf{w}_t$ are sampled from any general fixed sub-Gaussian. We assume bounded distribution for simplicity of exposition. Finally, we assume that the observed samples, $\dot{\mathbf{x}}_t$, are erased versions of $\mathbf{x}_t$, where for each entry j, independently,

$$\dot{\mathbf{x}}_t(j) = \begin{cases} \mathbf{x}_t(j) & w.p., \quad \delta \\ 0, & otherwise \end{cases}. \tag{4.2}$$

Hence, each sample, has δp observed entries in expectation.

Objective and Metric. Given a data stream $\{\mathbf{x}_1, \mathbf{x}_2, \dots\}$, the standard goal of streaming PCA is to recover the variance-maximizing subspace, i.e., of maximizing the *explained variance*. However, for our generative model, this corresponds to recovering the subspace spanned by the orthonormal matrix U. Now, we measure the error in estimation of the required subspace using

the largest principle angle based distance ([76]). That is, given any unitary matrix Q, we use the following distance:

$$\begin{aligned}\operatorname{dist}(U, Q) =& \operatorname{dist}\left(\operatorname{span}(U), \operatorname{span}(Q)\right) \\ =& \|U_\perp^\top Q\|_2 = \|Q_\perp^\top U\|_2 \end{aligned} \tag{4.3}$$

The distance is symmetric and takes values in $[0, 1]$. Given enough samples, the minimization of (4.3) is equivalent to maximizing the explained variance. In our experiments, Section 4.4, we use both metrics.

4.2 Algorithm

We now present our algorithm (see Algorithm 2) for the problem of streaming PCA, which uses some ideas from Algorithm 1. At a high level, the algorithm essentially leverages concentration of the sample covariance to the true covariance (Theorem 18) to estimate the next iterate.

Algorithm 2, takes in the stream of data vectors $\mathbf{x}_i$, the (known) probability of observation δ, the number of components k, and a block size B. It starts with a given k-dimensional subspace and refines that estimate doing a single pass over the data. Every subset of B subsequent samples is considered a block, even though only one sample is held in memory at any time.

To see why this algorithm works, consider line 7 of the algorithm and

over the course of block τ:

$$\begin{aligned} S_\tau =& \frac{1}{B} \sum_{t=B(\tau-1)+1}^{B\tau} \left[\frac{1}{\delta^2} \mathbf{x}_t \mathbf{x}_t^\top + \left(\frac{1}{\delta} - \frac{1}{\delta^2} \right) D_t \right] Q_{\tau-1} \\ =& \left[\frac{1}{B} \sum_{t=B(\tau-1)+1}^{B\tau} \frac{1}{\delta^2} \mathbf{x}_t \mathbf{x}_t^\top + \left(\frac{1}{\delta} - \frac{1}{\delta^2} \right) D_t \right] Q_{\tau-1} \\ =& \tilde{\Sigma}_B Q_{\tau-1}, \end{aligned}$$

where $D_t = \text{diag}(\mathbf{x}_t \mathbf{x}_t^\top)$. From the last line, we see that after every block, the algorithm is equivalent to performing a power iteration step. That is, the previous subspace estimate, $Q_{\tau-1}$, is essentially premultiplied by the estimator in (2.2) using all the samples in the block. The complication is that, with every block, the covariance estimate, $\tilde{\Sigma}_B$, is different. As we know from Chapter 3, this complicates the analysis requiring more advanced tools when compared to the simpler analysis of the classic power method.

It should be noted that, even though the algorithm effectively performs a power iteration per block, $\tilde{\Sigma}_n$ is never formed explicitly – all of the calculations can be performed in $O(kp)$ memory.

In Section 2.2 we discuss connections to other recent work, related to this problem and algorithm and in Section 4.3 we provide theoretical guarantees for the convergence of Algorithm 2.

4.3 Convergence Analysis

In this section we give theoretical guarantees for the convergence of Algorithm 2. In particular, we can show the following convergence result for

Algorithm 2

Input: $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$, δ, k, Block size: B, Starting Estimate: $H \in \mathbb{R}^{p\times k}$

$H \leftarrow Q_0 R_0$ (QR-decomposition)
for $\tau = 1, \ldots, n/B$ **do**
  $S_\tau \leftarrow 0$
  for $t = B(\tau-1)+1, \ldots, B\tau$ **do**
    $D_t \leftarrow \mathrm{diag}(\mathbf{x}_t \mathbf{x}_t^\top)$
    $S_\tau \leftarrow S_\tau + \frac{1}{B}\left[\frac{1}{\delta^2}\mathbf{x}_t\mathbf{x}_t^\top + \left(\frac{1}{\delta} - \frac{1}{\delta^2}\right) D_t\right] Q_{\tau-1}$
  end for
  $S_\tau = Q_\tau R_\tau$ (QR-decomposition)
end for
Return: Q_τ

Algorithm 2.

Theorem 16. *Consider a data stream, where $\dot{\mathbf{x}}_t \in \mathbb{R}^p$ for every t is generated by (4.1),(4.2), and the SVD of $A \in \mathbb{R}^{p\times k}$ is given by $A = U\Lambda V^\top$. Let, WLOG, $\lambda_1 = 1 \geq \lambda_2 \geq \cdots \geq \lambda_k > 0$. If $\sigma_k(U^T Q_0) \geq c$ for some constant c, then*

$$T = \Omega\left(\log(p/k\epsilon)/\log\left(\frac{\sigma^2 + 0.75\lambda_k^2}{\sigma^2 + 0.5\lambda_k^2}\right)\right),$$

$$B = \Omega\left(\frac{(k+\sigma^2)(k+\sigma^2 p)\log(2pT)}{\delta^2\epsilon^2\lambda_k^4}\right).$$

after T updates with block size B, dist$(U, Q_T) \leq \epsilon$, w.p. 0.99.

The result in Theorem 16 follows from the analysis in Section 3.4 and Theorem 18.

Assumption 17 (Sub-gaussian observations [47])**.** *The random vector $\mathbf{x} \in \mathbb{R}^p$ is sub-gaussian, that is $\|\mathbf{x}\|_{\psi_2} < \infty$. In addition, there exists a numerical constant $c_1 > 0$, such that: $\mathbb{E}(\langle \mathbf{x}, \mathbf{u}\rangle)^2 \geq c_1 \|\langle \mathbf{x}, \mathbf{u}\rangle\|_{\psi_2}^2$, $\forall \mathbf{u} \in \mathbb{R}^p$, where $\|\mathbf{x}\|_{\psi_2} = \inf\{u > 0 : \mathbb{E}\exp(|\mathbf{x}|^2/u^2) \leq 2\}$.*

Theorem 18 (Prop. 3, [47]). *Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$ be i.i.d. random vectors satisfying Assumption 17. Let $\mathbf{y}_1, \ldots, \mathbf{y}_n$ be the corresponding observed vectors with $\delta \in (0, 1]$. Then, for any $t > 0$, we have with probability at least $1 - e^{-t}$,*

$$\|\tilde{\Sigma}_n - \Sigma\|_2 \leq C \frac{\|\Sigma\|_2}{c_1} \cdot \max \left\{ \sqrt{\frac{\mathbf{r}(\Sigma)\,(t + \log(2p))}{\delta^2 n}}, \frac{\mathbf{r}(\Sigma)\,(t + \log(2p))}{\delta^2 n}\,(c_1\delta + t + \log n) \right\},$$

where $C > 0$ is an absolute constant and $\mathbf{r}(\Sigma) = \frac{\mathrm{tr}(\Sigma)}{\|\Sigma\|_2}$.

Theorem 19 (Theorem 1.4 of [69]). *Consider a finite sequence X_k of independent, random, self-adjoint matrices with dimension d. Assume that each random matrix satisfies $\mathbb{E}[X_k] = 0$ and $\|X_k\|_2 \leq R$ almost surely. Then, for all $t \geq 0$,*

$$Pr(\|\sum_k X_k\|_2 \geq t) \leq d \cdot \exp\left(\frac{-t^2/2}{\sigma^2 + Rt/3}\right),$$

where $\sigma^2 = \|\sum_k \mathbb{E}[X_k X_k^T]\|_2$.

4.4 Experiments

In this section, we perform a number of experiments that corroborate our theoretical claims and provide evidence that Algorithm 2 can perform better than the state of the art in several important regions. We start with describing the algorithms used in the experiments along with any implementation considerations. Then we proceed to experiments with synthesized, artificially sparsified real data, and naturally sparse data. For all these cases we compare the algorithms based on several performance metrics and discuss

their running times and robustness to parametrization. Since all the data sets are, of course, stored, we simulate the streaming and no-storage aspect for our algorithm.

Algorithm 2: The algorithm we propose. Reworking the equations on the number and size of blocks from Theorem 16 we can get an expression for T (the number of blocks) as a function of all given parameters. One important missing quantity is the ratio of eigenvalues at the cutoff point *which we do not assume we know*. For all the experiments that follow we use the following simplified formula:

$$T = C_{\text{Algo2}} \log \frac{pn\delta}{k}. \tag{4.4}$$

All of the parameters in the formula, are available before the start of the experiment, except for the erasure probability δ which can be very quickly and accurately estimated from the data stream, much faster than the PCA procedure itself. For all of our experiments in this manuscript, we use the constant $C_{Algo2} = \frac{1}{4}$ (see Section 4.4.4) and round the result to the nearest integer to get the number of blocks.

Stochastic Approximation: The most popular manifestation of Stochastic Approximation for PCA is Oja's rule ([55]). Even though it is not designed to deal with missing data, we nonetheless include it in our experiments as it is an industry standard. With every new sample ($\dot{\mathbf{x}}_t$) received, the algorithm

updates its estimate based on the following rule.

$$\tilde{U}_t = U_{t-1} + \frac{C_{SA}}{t}\dot{\mathbf{x}}_t\dot{\mathbf{x}}_t^T U_{t-1} \tag{4.5}$$

After each step, the intermediate estimate, $\tilde{U}_{t+1}$ is orthonormalized to give U_t.

The $O(\frac{1}{t})$ rule for the step size is accepted universally – see [55] and [9] for some discussion. However, to the best of our knowledge, the only complete characterization of the constant depends on the unknown eigengap at the cut-off point. For our experiments, we resort to picking a different constant C as suited to different datasets, as summarized in Section 4.4.4.

GROUSE: We include GROUSE in our experiments for it is a lightweight, fast and efficient algorithm, having proven to do well in most situations. For use in our experiments, we download the GROUSE Matlab code from the author's website. To make the algorithm well-defined in the region $k > \delta p$ (see discussion in Section 2.2), we make sure to use the pseudo-inverse operator for the projection step in (2.3). GROUSE is more complicated than Stochastic Approximation (see [11], or Section 2.2 for more references). The two algorithms, however, share a diminishing step-size, $\frac{C}{t}$. Again, we are faced with selecting a constant C_{GROUSE} and, much like Stochastic Approximation, there is no formula we can use in all cases. As we discuss in Subsection 4.4.4, we resort to using an individually tuned constant for every dataset.

Batch: As a simple – but not necessarily optimal – baseline for our experiments, we use the unbiased covariance estimator described in (2.2). This is

computed bringing all the samples in memory at once, hence the characterization "batch." We only include it in our first few experiments for validation purposes. It is omitted in the larger, real datasets as it is the most resource intensive of all algorithms considered here.

4.4.1 Simulations on the model

We start our experiments in a fully controlled setting. For that, we synthesize data points based on the model at (4.2). While this is a fairly general model, we widen our scope to real datasets in the remainder on this section. Figure 4.1 demonstrates a single example run for a case when the number of observed entries per sample is smaller than the target number of principal components. Figure 4.2 shows another example run with a single, highly coherent component. Methods using a single sample to update seem to be having trouble. We see this behaviour again in Figure 4.4.

Single-run convergence figures give us a good understanding of how things look, but are by no means evidence of a trend. To demonstrate the performance of all algorithms, we perform many independent runs in several diverse scenaria and present the averages.

Figure 4.3 showcases a qualitative difference between the studied algorithms. We study the transition from the region where $k < \delta p$ (more observed entries than components) to $k > \delta p$, or the *no-completion region*. Notice that the performance of Algorithm 2 deteriorates gracefully. On the other hand, imputation-based algorithms (like GROUSE) are ill-defined in that region (as

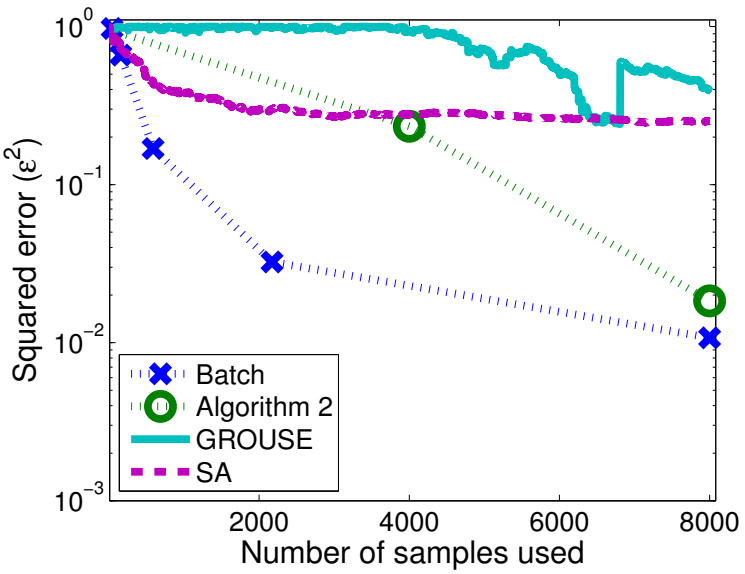


Figure 4.1: Example convergence curve with fewer observed entries than rank on average ($p = 20$, $k = 5$, $\delta = 0.2$, $\sigma = 0.2$).

discussed in Section 2.2) and show rapid deterioration in performance.

In Figure 4.4 we study the dependence of performance on the coherence of the signal components (spikes). Most algorithms show a gradual deterioration as the component becomes more coherent, with the exception of the Stochastic Approximation algorithm.

4.4.2 Gas Sensor Array Data

For our first experiment with real data we use the gas sensor array drift dataset from [71]. It consists of 13910 samples with 128 entries each, all measurements of gas concentrations. The dataset has no missing entries and

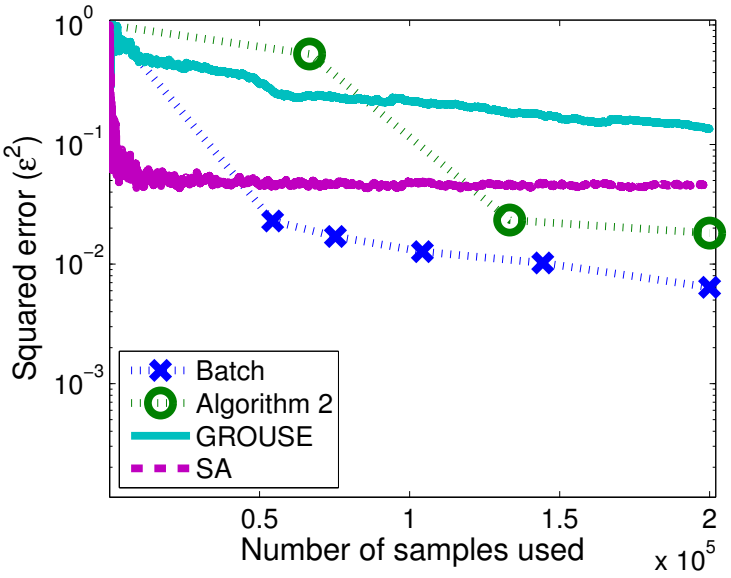


Figure 4.2: Example convergence curves for one highly coherent component ($p = 100$, $k = 1$, $\delta = 0.05$, $\mu = 0.95p$).

we use it as an intermediate step between synthetic and real data as follows: First we randomly permute its samples. Then we consider the samples in order, and simulate our erasure model from Section 4.1. That is, every entry is observed independently with probability δ. Unobserved entries are replaced with zeros. We do a predetermined number of passes over the whole dataset before reporting the final performance. To evaluate performance we use the classic metric of explained variance. Let X denote a matrix containing all samples, and let $Q \in \mathbf{R}^{p \times k}$ denote the subspace estimate provided by the algorithm. The metric of *explained variance*, is given by $||Q^T X||_F$, which we normalize with $||X||_F$ to bring into the $[0, 1]$ range.

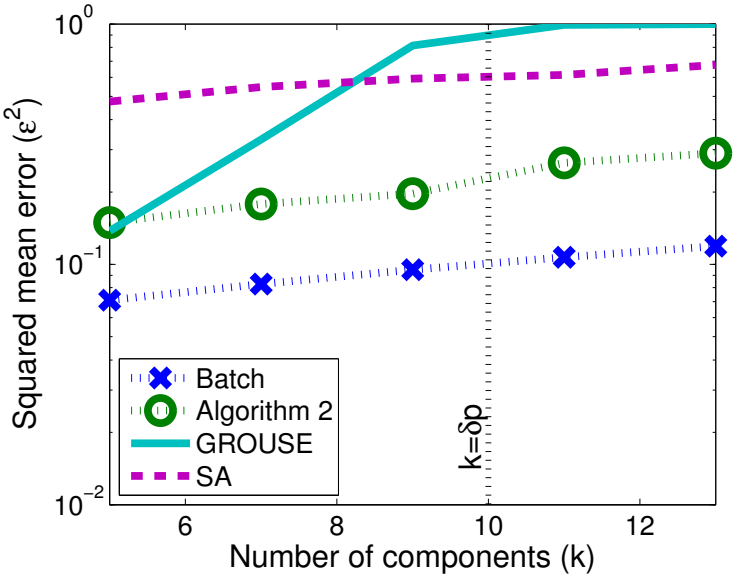


Figure 4.3: Transition around the boundary $k = \delta p$ ($p = 100$, $\delta = 0.01$, $\sigma = 0.2$, average of 134 runs)

In Figure 4.5, we see that Algorithm 2 is able to achieve maximal explained variance, while being more robust with respect to the choice of k. To compare the running times of the 3 algorithms we calculate the average running time in seconds per sample and report these times in Table 4.1.

4.4.3 MovieLens

In our last set of experiments, we use the MovieLens dataset from [31]. It contains about 10 million ratings for 10 thousand movies by 72 thousand users of the MovieLens service. The dataset is naturally sparse: every user only rates a tiny fraction of the movies in the database.

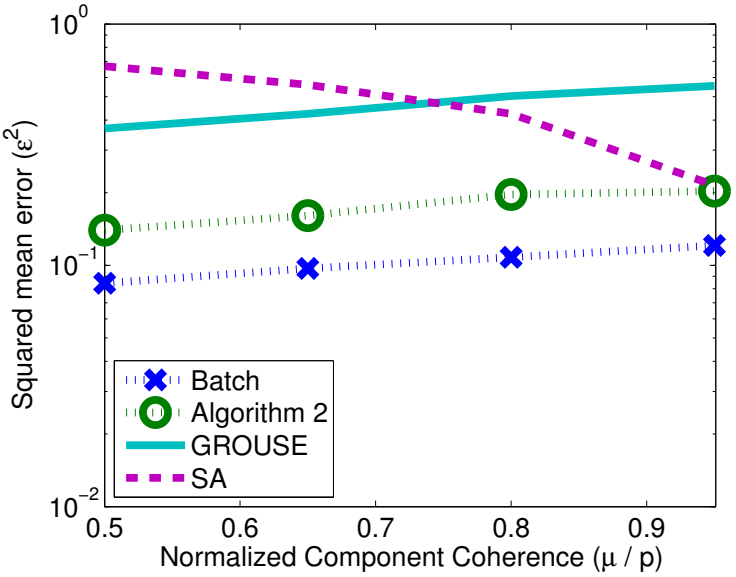


Figure 4.4: Performance vs coherence of the signal component ($p = 100$, $\delta = 0.05$, $\sigma = 0.2$, average of 56 runs)

In this case again, there is no access to the "true" principal components, so instead of the distance metric in (4.3), we evaluate based on the explained variance.

To separate training from testing, we adhere to the following procedure: We first split the 10M ratings in the dataset into training and testing sets, with a 70/30 ratio. The training ratings are fed into the algorithms; each user is considered as a sample. Finally, let M_{test} denote the testing set in matrix form (movies by users) and let $Q \in \mathbf{R}^{p \times k}$ denote the subspace output by the algorithm. We evaluate based on the normalized explained variance, given by $||Q^T M_{test}||_F / ||M_{test}||_F$.

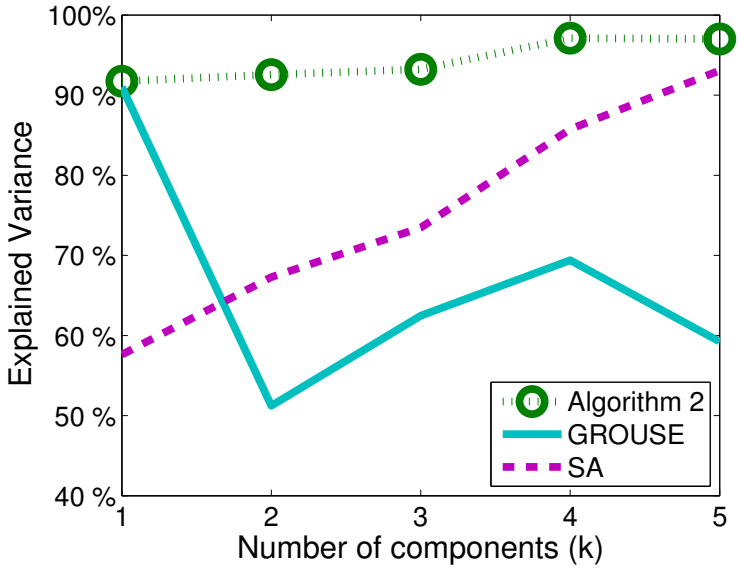


Figure 4.5: Performance on the gas sensor array dataset ($\delta = 0.02$, 30 independent passes)

In Figure 4.6 we see that, after *only a single pass over the dataset*, our algorithm is able to explain almost as much variance, as the batch algorithm and achieve a significant gap over GROUSE and improve over SA. The running times for this experiment are reported in Table 4.1.

4.4.4 Ease of parametrization

As discussed in this section, a common theme for GROUSE and SA is the choice of the constant used in the step-size sequence. This can prove to be a very time-consuming task, especially in the case when no ground truth information is available. A "good" constant for one experiment might be

Table 4.1: Average running time per processed sample

Experiment		**Algorithm 2**	**GROUSE**	**SA**
Gas	**k=1**	1.049e-04	1.312e-04	5.274e-05
	k=2	1.027e-04	1.306e-04	4.587e-05
	k=3	1.094e-04	1.521e-04	5.654e-05
	k=4	9.666e-05	1.347e-04	4.870e-05
	k=5	1.157e-04	1.681e-04	6.372e-05
ML	**k=5**	4.617e-02	1.796e-01	3.179e-01
	k=10	3.854e-02	3.138e-02	3.078e-02
	k=15	5.075e-02	3.711e-02	3.210e-01
	k=20	5.530e-02	6.148e-02	7.646e-02

completely unsuitable for another. This forces us to look for a good parameter in every experiment we run. We went to great lengths to pick an ideal constant every time – still it is likely that slightly better choices exist. Such is the nature of this endeavor.

To demonstrate the complexity we are faced with in our experiments and to enable reproducibility of our results, we corral the values that were used in our real-data experiments and present them in Table 4.2. An important feature of Algorithm 2 that we want to emphasize here is that we were able to use *a single parameter for all of our experiments*. This makes Algorithm 2 very appealing for deployment on new datasets.

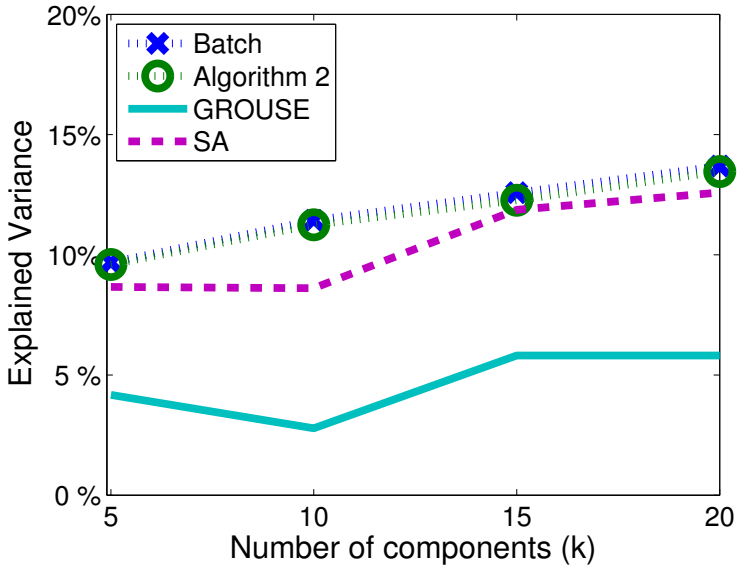


Figure 4.6: Performance on the MovieLens dataset

Table 4.2: Parametrization for real data

Experiment		C_{Algo2}	C_{GROUSE}	C_{SA}
Gas	k=1	**0.25**	2.644e-06	3.944e-06
	k=2	**0.25**	1.261e-05	1.881e-05
	k=3	**0.25**	3.158e-05	4.711e-05
	k=4	**0.25**	1.216e-04	1.813e-04
	k=5	**0.25**	2.861e-04	4.269e-04
ML	k=5	**0.25**	5.265e+00	6.582e+00
	k=10	**0.25**	2.315e+01	2.893e+01
	k=15	**0.25**	2.569e+00	3.854e+00
	k=20	**0.25**	4.652e+00	6.978e+00

Chapter 5

Fast PageRank Approximations on Graph Engines

In this chapter, we propose FrogWild!, a novel algorithm for fast approximation of high PageRank vertices It is geared towards reducing network costs of running traditional PageRank algorithms. Our algorithm can be seen as a quantized version of power iteration that performs multiple parallel random walks over a directed graph. One important innovation is that we introduce a modification to the GraphLab framework that only partially synchronizes mirror vertices. This partial synchronization vastly reduces the network traffic generated by traditional PageRank algorithms, thus greatly reducing the per-iteration cost of PageRank. On the other hand, this partial synchronization also creates dependencies between the random walks used to estimate PageRank. Our main theoretical innovation is the analysis of the correlations introduced by this partial synchronization process and a bound establishing that our approximation is close to the true PageRank vector.

We implement our algorithm in GraphLab and compare it against the default PageRank implementation. We show that our algorithm is very fast, performing each iteration in less than one second on the Twitter graph and

can be up to 7× faster compared to the standard GraphLab PageRank implementation.

Contributions: We consider the problem of fast and efficient (in the sense of time, computation and communication costs) computation of the high PageRank nodes, using a graph engine. To accomplish this we propose and analyze a new PageRank algorithm specifically designed for the graph engine framework and, significantly, we propose a modification of the standard primitives of the graph engine framework (in particular, GraphLab PowerGraph), that enables significant network savings. We explain in further detail both our objectives, and our key innovations.

Rather than seek to recover the full PageRank vector, we aim for the top k PageRank vertices (where k is considered to be approximately in the order of $10 - 1000$). Given an output of a list of k vertices, we define two natural accuracy metrics that compare the true top-k list with our output. The algorithm we propose, FrogWild, operates by starting a small (sublinear in the number of vertices n) number of random walkers (*frogs*) that jump randomly on the directed graph. The random walk interpretation of PageRank enables the frogs to jump to a completely random vertex (teleport) with some constant probability (set to 0.15 in our experiments, following standard convention). After we allow the frogs to jump for time equal to the mixing time of this non-reversible Markov chain, their positions are sampled from the invariant distribution π which is the PageRank vector. The standard PageRank iteration can be seen as the continuous limit of this process (*i.e.*, the frogs

become water), which is equivalent to power iteration for stochastic matrices.

The main *algorithmic contributions* of this work are comprised of the following three innovations. First, we argue that discrete frogs (a quantized form of power iteration) is significantly better for distributed computation, when one is interested only in the large entries of the vector π. This is because each frog produces an independent sample from π. If some entries of π are substantially larger and we only want to determine those, a small number of independent samples suffices. We make this formal using standard Chernoff bounds (see also [64, 25] for similar arguments). On the contrary, during standard PageRank iterations, vertices pass messages to all their out-neighbors since a non-zero amount of water must be transferred. This tremendously increases the network bandwidth especially when the graph engine is over a cluster with many machines.

One major issue with simulating discrete frogs on a graph engine is teleportations. Graph frameworks partition vertices to physical nodes and restrict communication on the edges of the underlying graph. Global random jumps would create dense messaging patterns that would increase communication. Our second innovation is a way of obtaining an identical sampling behavior without teleportations. We achieve this by initiating the frogs at uniformly random positions and having them perform random walks for a life span that follows a geometric random variable. The geometric probability distribution depends on the teleportation probability and can be calculated explicitly.

Our third innovation involves a simple proposed modification for graph

frameworks. Most modern graph engines (like GraphLab PowerGraph [30]) employ vertex-cuts as opposed to edge-cuts. This means that each vertex of the graph is assigned to multiple machines so that graph edges see a local vertex mirror. One copy is assigned to be the master and maintains the master version of the vertex data while remaining replicas are mirrors that maintain local cached read–only copies of the data. Changes to the vertex data are made to the master and then replicated to all mirrors at certain synchronization barriers. This architecture is highly suitable for graphs with high-degree vertices (as most real-world graphs are) but has one limitation when used for a few random walks: imagine that vertex v_1 contains one frog that wants to jump to v_2. If vertex v_1 has very high degree, it is very likely that multiple replicas of that vertex exist, possibly one in each machine in the cluster. In an edge-cut scenario only one message would travel from $v_1 \rightarrow v_2$, assuming v_1 and v_2 are located in different physical nodes. However, when vertex-cuts are used, the state of v_1 is updated (i.e., contains no frogs now) and this needs to be communicated to all mirrors. It is therefore possible that *a single random walk can create a number of messages equal to the number of machines in the cluster.*

We modify PowerGraph to expose a scalar parameter p_s per vertex. By default, when the framework is running, in each super-step all masters synchronize their programs and vertex data with their mirrors. Our modification is that for each mirror we flip an independent coin and synchronize with probability p_s. Note that when the master does not synchronize the vertex

program with a replica, that replica will not be active during that super-step. Therefore, we can avoid the communication and CPU execution by performing limited synchronization in a randomized way.

FrogWild! is therefore executed asynchronously but relies on the Bulk Synchronous execution mode of PowerGraph with the additional simple randomization we explained. The name of our algorithm is inspired by HogWild [59], a lock-free asynchronous stochastic gradient descent algorithm proposed by Niu *et al.*. We note that PowerGraph does support an asynchronous execution mode [30] but we implemented our algorithm by a small modification of synchronous execution. As discussed in [30], the design of asynchronous graph algorithms is highly nontrivial and involves locking protocols and other complications. Our suggestion is that for the specific problem of simulating multiple random walks on a graph, simply randomizing synchronization can give significant benefits while keeping design simple.

While the parameter p_s clearly has the power to significantly reduce network traffic – and indeed, this is precisely born out by our empirical results – it comes at a cost: the standard analysis of the Power Method iteration no longer applies. The main challenge that arises is the theoretical analysis of the FrogWild algorithm. The model is that each vertex is separated across machines and each connection between two vertex copies is present with probability p_s. A single frog performing a random walk on this new graph defines a new Markov Chain and this can be easily designed to have the same invariant distribution π equal to normalized PageRank. The complication is that the

trajectories of frogs are *no longer independent*: if two frogs are in vertex v_1 and (say) only one mirror v_1' synchronizes, both frogs will need to jump through edges connected with that particular mirror. Worse still, this correlation effect increases, the more we seek to improve network traffic by further decreasing p_s. Therefore, it is no longer true that one obtains independent samples from the invariant distribution π. Our theoretical contribution is the development of an analytical bound that shows that these dependent random walks still can be used to obtain $\hat{\pi}$ that is provably close to π with high probability. We rely on a coupling argument combined with an analysis of pairwise intersection probabilities for random walks on graphs. In our convergence analysis we use the *contrast bound* [20] for non-reversible chains.

We now make precise the intuition and outline given in the introduction. Given the definition of PageRank in Section 2.6, we first define the problem of approximating the top elements. We then define the algorithm, state our main analytical results and provide full analysis. We conclude with a comprehensive set of experiments.

5.1 Top PageRank Elements

Given the true PageRank vector, π and an estimate given by an approximate PageRank algorithm, we define the estimate's top-k accuracy using one of two metrics.

Definition 20 (Mass Captured)**.** *Given distribution $v \in \Delta^{n-1}$, the true PageRank distribution $\pi \in \Delta^{n-1}$ and an integer $k \geq 0$, we*

define the mass captured *by v as follows.*

$$\mu_k(v) \triangleq \pi(\text{argmax}_{|S|=k} v(S))$$

For a set $S \subset [n]$, $v(S) \triangleq \sum_{i \in S} v(i)$ denotes the total mass ascribed to the set by the distribution $v \in \Delta^{n-1}$.

Put simply, the set S^* that gets the most mass according to v out of all sets of size k, is evaluated according to π and that gives us our metric. It is maximized by π itself, i.e. the optimal value is $\mu_k(\pi)$.

The second metric we use is the exact identification probability, *i.e.* the fraction of elements in the output list that are also on the true top-k list. Note that the second metric is limited in that it does not give partial credit for high PageRank vertices that are not on the true top-k list. In our experiments in Section 5.5, we mostly use the normalized captured mass accuracy metric but also report the exact identification probability for some cases – typically the results are similar.

Our algorithm approximates the heaviest elements of the invariant distribution of a Markov Chain, by simultaneously performing multiple random walks on the graph. The main modification to PowerGraph is the exposure of a parameter, p_s, controlling the probability that a given master node synchronizes with any one of its mirrors. Per step, this leads to a proportional reduction in network traffic. The main contribution of this work is to show that we get results of comparable or improved accuracy, while maintaining this

network traffic advantage. We demonstrate this analytically in Section 5.3 and empirically in Section 5.5.

5.2 Algorithm

During setup, the graph is partitioned using GraphLab's default ingress algorithm. At this point each one of N frogs is born on a vertex chosen uniformly at random. Each vertex i carries a counter initially set to 0 and denoted by $c(i)$. Scheduled vertices execute the following program.

Incoming frogs from previously executed vertex programs, are collected by the `init()` function. At `apply()` every frog dies with probability $p_T = 0.15$. This, along with a uniform starting position, effectively simulates the 15% uniform component from Definition 1.

A crucial part of our algorithm is the change in synchronization behaviour. The <`sync`> step only synchronizes a p_s fraction of mirrors leading to commensurate gains in network traffic (cf. Section 5.5). This patch on the GraphLab codebase was only a few lines of code. Section 5.5 contains more details regarding the implementation.

The `scatter()` phase is only executed for edges e incident to a mirror of i that has been synchronized. Those edges draw a binomial number of frogs to send to their other endpoint. The rest of the edges perform no computation. The frogs sent to vertex j at the last step will be collected at the `init()` step when j executes.

Parameter p_T is the teleportation probability from the random surfer model in [56]. To get PageRank using random walks, one could adjust the transition matrix P as described in Definition 1 to get the matrix Q. Alternatively, the process can be replicated by a random walk following the original matrix P, and teleporting at every time, with probability p_T. The destination for this teleportation is chosen uniformly at random from $[n]$. We are interested in the position of a walk at a predetermined point in time as that would give us a sample from π. This holds as long as we allow enough time for mixing to occur.

Due to the inherent markovianity in this process, one could just consider it starting from the last teleportation before the predetermined stopping time. When the stopping time is late enough, the number of steps performed between the last teleportation and the predetermined stopping time, denoted by X, is geometrically distributed with parameter p_T. This follows from the time-reversibility in the teleportation process: inter-teleportation times are geometrically distributed, so as long as the first teleportation event happens before the stopping time, then $X \sim \text{Geom}(\text{p}_\text{T})$.

This establishes that, the FrogWild! process – where a frog performs a geometrically distributed number of steps following the original transition matrix P – closely mimics a random walk that follows the adjusted transition matrix, Q. In practice, we stop the process after t steps to get a good approximation. To show our main result, Theorem 24, we analyze the latter process.

FrogWild! vertex program

Input parameters: p_s, $p_T = 0.15$, t

apply(i) $K(i) \leftarrow$ [# incoming frogs]

If t steps have been performed:

$c(i) \leftarrow c(i) + K(i)$
HALT

For every incoming frog:

With probability p_T, frog dies:
$c(i) \leftarrow c(i) + 1$,
$K(i) \leftarrow K(i) - 1$.

<**sync**> For every *mirror* m of vertex i:

With probability p_s:

Synchronize state with mirror m.

scatter($e = (i, j)$) [Only on synchronized mirrors]

Generate Binomial number of frogs:

$$x \sim Bin\left(K(i), \frac{1}{d_{\text{out}}(i)p_s}\right)$$

Send x frogs to vertex j: `signal(j,x)`

Using a binomial distribution to independently generate the number of frogs in the `scatter()` phase closely models the effect of random walks. The marginal distributions are correct, and the number of frogs, that did not die during the `apply()` step, is preserved in expectation. For our implementation

we resort to a more efficient approach. Assuming $K(i)$ frogs survived the `apply()` step, and M mirrors where picked for synchronization, then we send $\lceil \frac{K(i)}{M} \rceil$ frogs to $\min(K(i), M)$ mirrors. If the number of available frogs is less than the number of synchronized mirrors, we pick $K(i)$ arbitrarily.

5.3 Main Result

Our analytical results essentially provide a high probability guarantee that our algorithm produces a solution that approximates well the PageRank vector. Recall that the main modification of our algorithm involves randomizing the synchronization between master nodes and mirrors. For our analysis, we introduce, in Section 5.6.1, a broad model to deal with partial synchronization.

Our results tell us that partial synchronization does not change the distribution of a single random walk. To our statements clear, we need the simple definition. Note that for a time-varying vector x, in this chapter, we denote its value at time t by x^t.

Definition 21. *We denote the state of random walk i at its t^{th} step by s_i^t.*

Then, we see that $\mathbb{P}\big(s_1^{t+1} = i \big| s_1^t = j\big) = 1/d_{\text{out}}(j)$, and $x_1^{t+1} = Px_1^t$. This follows simply by the symmetry assumed in Definition 27. Thus if we were to sample serially, the modification of the algorithm controlling (limiting) synchronization would not affect each sample, and hence would not affect our estimate of the invariant distribution. However, we start multiple (all) random

walks simultaneously. In this setting, the fundamental analytical challenge stems from the fact that random walks that intersect are correlated. The key to our result is that we can control the effect of this correlation, as a function the parameter p_s and the *pairwise probability* that two random walks intersect. We define this formally.

Definition 22. *Suppose two walkers l_1 and l_2 start at the same time and perform t steps. The probability that they meet is defined as follows.*

$$p_{\cap}(t) \triangleq \mathbb{P}\left(\exists\ \tau \in [0,t],\ \text{s.t. } s_{l_1}^{\tau} = s_{l_2}^{\tau}\right) \tag{5.1}$$

Definition 23 (Estimator)**.** *Given the positions of N random walks at stopping time t, $\{s_l^t\}_{l=1}^N$, we define the following estimator for the invariant distribution π.*

$$\hat{\pi}_N(i) \triangleq \frac{|\{l : l \in [N], s_l^t = i\}|}{N} = \frac{c(i)}{N} \tag{5.2}$$

Here $c(i)$ refers to the tally maintained by the FrogWild! *vertex program.*

Now we can state the main result. Here we give a guarantee for the quality of the solution furnished by our algorithm.

Theorem 24 (Main Theorem)**.** *Consider N frogs following the* FrogWild! *process (Section 5.2), under the erasure model of Definition 27. The frogs start at independent locations, distributed uniformly and stop after a geometric number of steps or, at most, t steps. The estimator $\hat{\pi}_N$ (Definition 23), captures mass close to the optimal. Specifically, with probability at least $1-\delta$,*

$$\mu_k(\hat{\pi}_N) \geq \mu_k(\pi) - \epsilon,$$

where

$$\epsilon < \sqrt{\frac{(1-p_T)^{t+1}}{p_T}} + \sqrt{\frac{k}{\delta}\left[\frac{1}{N} + (1-p_s^2)p_\cap(t)\right]}. \tag{5.3}$$

Remark 1 (Scaling)**.** *The result in Theorem 24 immediately implies the following scaling for the number of iterations and frogs respectively. They both depend on the maximum captured mass possible, $\mu_k(\pi)$ and are sufficient for making the error, ϵ, of the same order as $\mu_k(\pi)$.*

$$t = O\left(\log \frac{1}{\mu_k(\pi)}\right), \qquad N = O\left(\frac{k}{\mu_k(\pi)^2}\right)$$

The proof of Theorem 24 is deferred to Section 5.6.2. The guaranteed accuracy via this result also depends on the probability that two walkers will intersect. Via a simple argument, that probability is the same as the meeting probability for independent walks. The next theorem bounds this probability.

Theorem 25 (Intersection Probability)**.** *Consider two independent random walks obeying the same ergodic transition probability matrix, Q with invariant distribution π, as described in Definition 1. Furthermore, assume that both of them are initially distributed uniformly over the state space of size n. The probability that they meet within t steps, is bounded as follows,*

$$p_\cap(t) \leq \frac{1}{n} + \frac{t\|\pi\|_\infty}{p_T},$$

where $\|\pi\|_\infty$, denotes the maximal element of the vector π.

The proof is based on the observation that the l_∞ norm of a distribution controls the probability that two independent samples coincide. We show that

for all steps of the random walk, that norm is controlled by the l_∞ norm of π. We defer the full proof to Section 5.6.3.

A number of studies, give experimental evidence (e.g. [14]) suggesting that PageRank values for the web graph follow a power-law distribution with parameter approximately $\theta = 2.2$. That is true for the tail of the distribution – the largest values, hence of interest to us here – regardless of the choice of p_T. The following proposition bounds the value of the heaviest PageRank value, $\|\pi\|_\infty$.

Proposition 26 (Maximum of Power-Law)**.** *Let $\pi \in \Delta^{n-1}$ follow a power-law distribution with parameter θ and minimum value p_T/n. Its maximum element, $\|\pi\|_\infty$, is at most $n^{-\gamma}$, with probability at least $1 - cn^{\gamma - \frac{1}{\theta-1}}$, for some universal constant c.*

Proof. The expected maximum value of n independent draws from a power-law distribution with parameter θ, is shown in [54] to be

$$\mathbb{E}x_{max} = O(n^{-\frac{1}{\theta-1}}).$$

Simple application of Markov's inequality, gives us the statement. □

Assuming $\theta = 2.2$ and picking, for example, $\gamma = 0.5$, we get

$$\mathbb{P}(\|\pi\|_\infty > 1/\sqrt{n}) \leq cn^{-1/3}.$$

This implies that with probability at least $1 - cn^{-1/3}$ the meeting probability is bounded as follows.

$$p_\cap(t) \leq \frac{1}{n} + \frac{t}{p_T\sqrt{n}}.$$

One would usually take a number of steps t that are either constant or logarithmic with respect to the graph size n. This implies that for many reasonable choices of set size k and acceptable probability of failure δ, the meeting probability vanishes as n grows. Then we can make the second term of the error in (5.3) arbitrarily small by controlling the number of frogs, N.

5.4 Related Work

There is a very large body of work on computing and approximating PageRank on different computation models (*e.g.* see [16, 21, 64, 25, 4] and references therein). To the best of our knowledge, our work is the first to specifically design an approximation algorithm for high-PageRank nodes for graph engines. Another line of work looks for *Personalized PageRank* (PPR) scores. This quantifies the influence an arbitrary node i has on another node j, cf. recent work [46] and discussion therein. In [8], the top-k approximation of PPR is studied. However, PPR is not applicable in our case, as we are looking for an answer close to a *global* optimum.

In [7], a random-walks-based algorithm is proposed. The authors provide some insightful analysis of different variations of the algorithm. They show that starting a single walker from every node, is sufficient to achieve a good global approximation. We focus on capturing a few nodes with a lot of mass, hence we can get away with orderwise much fewer frogs than $O(n)$. This is important for achieving low network traffic when the algorithm is executed on a distributed graph framework. Figure 5.10 shows linear reduction

in network traffic when the number of initial walkers decreases. Furthermore, our method does not require waiting for the last frog to naturally expire (note that the geometric distribution has infinite support). We impose a very short time cut-off, t, and exactly analyze the penalty in captured mass we pay for it in Theorem 24.

One natural question is how our algorithm compares to, or can be complemented by, graph sparsification techniques. One issue here is that graph sparsification crucially depends on the similarity metric used. Well-studied properties that are preserved by sparsification methods include lengths of shortest paths between vertices (such sparsifiers are called Spanners, see e.g. [57]), cuts between subsets of vertices [15] and more generally quadratic forms of the graph Laplacian [67, 13], see [13] and references therein for a recent overview. To the best of our knowledge, there are no known graph sparsification techniques that preserve vertex PageRank.

One natural heuristic that one may consider is to independently flip a coin and delete each edge of the graph with some probability r. Note that this is crucially different from spectral sparsifiers [67, 13] that choose these probabilities using a process that is already more complicated than estimating PageRank. This simple heuristic of independently deleting edges indeed accelerates the estimation process for high-PageRank vertices. We compare FrogWild to this uniform sparsification process in Figure 5.6. We present here results for 2 iterations of the GraphLab PR on the sparsified graph. Note that running only one iteration is not interesting since it actually esti-

mates only the in-degree of a node which is known in advance (i.e., just after the graph loading) in a graph engine framework. It can be seen in Figure 5.6 that even when only two iterations are used on the sparsified graph the running time is significantly worse compared to FrogWild and the accuracy is comparable.

Our base-line comparisons come from the graph framework papers since PageRank is a standard benchmark for running-time, network and other computations. Our implementation is on GraphLab (PowerGraph) and significantly outperforms the built-in PageRank algorithm. This algorithm is already shown in [30, 65] to be significantly more efficient compared to other frameworks like Hadoop, Spark, Giraph *etc.*

5.5 Experiments

In this section we compare the performance of our algorithm to the PageRank algorithm shipped with GraphLab v2.2 (PowerGraph) [48]. The fact that GraphLab is the fastest distributed engine for PageRank is established experimentally in [65]. We focus on two algorithms: the basic built-in algorithm provided as part of the GraphLab *graph analytics toolkit*, referred to here as GraphLab PR, and FrogWild. Since we are looking for a top-k approximation and GraphLab PR is meant to find the entire PageRank vector, we only run it for a small number of iterations (usually 2 are sufficient). This gives us a good top-k approximation and is much faster than running the algorithm until convergence. We also fine tune the algorithm's tolerance

parameter to get a good but fast approximation.

We compare several performance metrics, namely: running time, network usage, and accuracy. The metrics do not include time and network usage required for loading the graph into GraphLab (known as the *ingress time*). They reflect only the execution stage.

5.5.1 The Systems

We perform experiments on two systems. The first system is a cluster of 20 virtual machines, created using VirtualBox 4.3 [74] on a single physical server. The server is based on an Intel® Xeon® CPU E5-1620 with 4 cores at 3.6 GHz, and 16 GB of RAM. The second system, comprises of a cluster of up to 24 EC2 machines on AWS (Amazon web services) [3]. We use m3.xlarge instances, based on Intel® Xeon® CPU E5-2670 with 4 vCPUs and 15 GB RAM.

5.5.2 The Data

For the VirtualBox system, we use the LiveJournal graph [44] with 4.8M vertices and 69M edges. For the AWS system, in addition to the LiveJournal graph, we use the Twitter graph [42] which has 41.6M nodes and 1.4B edges.

5.5.3 Implementation

FrogWild is implemented on the standard GAS (gather, apply, scatter) model. We implement `init()`, `apply()`, and `scatter()`. The purpose of

`init()` is to collect the random walks sent to the node by its neighbors using `scatter()` in the previous iteration. In the first iteration, `init()` generates a random fraction of the initial total number of walkers. This implies that the initial walker locations are randomly distributed across nodes. FrogWild requires the length of random walks to be geometrically distributed (see Section 5.2). For the sake of efficiency, we impose an upper bound on the length of random walks. The algorithm is executed for the constant number of iterations (experiments show good results with even 3 iterations) after which all the random walks are stopped simultaneously. The `apply()` function is responsible for keeping track of the number of walkers that have stopped on each vertex and `scatter()` distributes the walkers still alive to the neighbors of the vertex. The `scatter()` phase is the most challenging part of the implementation. In order to reduce information exchange between machines, we use a couple of ideas.

First, we notice that random walks do not have identity. Hence, random walks destined for the same neighbor can be combined into a single message. The main optimization and most significant part of our work is modifying the GraphLab engine to support *randomized synchronization*, as described in Section 5.2. We expose the synchronization probability $p_s \in [0, 1]$ to the user as a small extension to the GraphLab API. It describes the fraction of replicas that will be synchronized. Replicas not synchronized remain idle for the upcoming scatter phase. The source changes in the engine are a matter of a few (about 10) lines of code. Using this feature is completely optional;

i.e., setting $p_s = 1$ results in using the original engine, leaving other analytic workloads unaffected. However, any random walk or "gossip" style algorithm (that sends a single messages to a random subset of its neighbors) can benefit by reducing p_s. Our modification of the GraphLab engine as well as the FrogWild vertex program can be found in [17].

5.5.4 Results

FrogWild is significantly faster and uses less network and CPU compared to GraphLab PR. Let us start with the Twitter graph and the AWS system. In Figure 5.1(a) we see that, while GraphLab PR takes about 7.5 seconds per iteration (for 12 nodes), FrogWild takes less than 1 sec, achieving more than a $7x$ speedup. Reducing the value of p_s decreases the running time. We see a similar picture with the total running time of the algorithms in Figure 5.1(b)).

We plot network performance in Figure 5.2(c). We get a $1000x$ improvement compared to the exact GraphLab PR, and more than $10x$ with respect to doing 1 or 2 iterations of GraphLab PR. In Figure 5.2(d) we can see that the total CPU usage reported by the engine is also much lower for FrogWild.

We now turn to compare the approximation metrics for the PageRank algorithm. For various k, we check the two accuracy metrics: Mass captured (Figure 5.3(a)) and the Exact identification (Figure 5.3(b)). Mass captured – is the total PageRank that the reported top-k vertices worth in the exact

ranking. Exact identification – is the number of vertices in the intersection of the reported top-k and the exact top-k lists. We can see that the approximation achieved by the FrogWild for $p_s = 1$ and $p_s = 0.7$ always outperforms the GraphLab PR with 1 iteration. The approximation achieved by the FrogWild with $p_s = 0.4$ is relatively good for both metrics, and with $p_s = 0.1$ is reasonable for *mass captured*.

In Figure 5.4 we can see the trade-off between the accuracy, total running time, and the network usage. The performance of FrogWild is evaluated for various number of iterations and the values of p_s. The results show that with the accuracy comparable to GraphLab PR, FrogWild has much less running time and network usage. Figure 5.5 illustrates how much network traffic we save using FrogWild. The area of each circle is proportional to the number of bytes sent by each algorithm.

We also compare FrogWild to an approximation strategy that uses a simple sparsification technique described in Section 5.4. First, the graph is sparsified by deleting each edge with probability r, then GraphLab PR is executed. In Figure 5.6, we can see that FrogWild outperforms this approach in terms running time while achieving comparable accuracy.

Finally, we plot results for the LiveJournal graph on the VirtualBox system. Figures 5.7(a,b) show the effect of the number of walkers, N, and the number of iterations for FrogWild on the achieved accuracy. Good accuracy and running time (see Figure 5.8(c,d)) are achieved for $800K$ initial random walks and 4 iterations of FrogWild. Similar to the Twitter graph, also for

the LiveJournal graph we can see, in Figure 5.9, that our algorithm is faster and uses much less network, while still maintaining good PageRank accuracy. By varying the number of initial random walks and the number of iterations we can fine-tune the FrogWild for the optimal accuracy-speed trade-off.

Interestingly, for both graphs (Twitter and LiveJournal), reasonable parameters are: 800K initial random walks and 4 iterations, despite the order of magnitude difference in the graph sizes. This implies slow growth for the necessary number of frogs with respect to the size of the graph. This scaling behavior is tough to check in practice, but it is explained by our analysis. Specifically, Remark 1 shows that the number of frogs should scale as $N = O\left(\frac{k}{\mu_k(\pi)^2}\right)$.

5.6 Analysis

In this section, we provide detailed proofs for all posed statements. First, we introduce an edge erasure model that provides both intuition and rigor for the subsequent analysis.

5.6.1 Edge Erasure Model

Definition 27 (Edge Erasure Model)**.** *An edge erasure model is a process that is independent from the random walks (up to time t) and temporarily erases a subset of all edges at time t. The event $E_{i,j}^t$ represents the erasure of edge (i, j) from the graph for time t. The edge is not permanently removed from the graph, it is just disabled and considered again in the next step. The edge*

erasure models we study satisfy the following properties.

1. *Edges are erased independently for different vertices,*

$$\mathbb{P}(E^t_{i,j}, E^t_{i,k}) = \mathbb{P}(E^t_{i,j})\mathbb{P}(E^t_{i,k})$$

   and across time,

$$\mathbb{P}(E^t_{i,j}, E^s_{i,j}) = \mathbb{P}(E^t_{i,j})\mathbb{P}(E^s_{i,j}).$$

2. *Each outgoing edge is preserved (not erased) with probability at least p_s.*

$$\mathbb{P}(\overline{E^t_{i,j}}) \geq p_s$$

3. *Erasures do not exhibit significant negative correlation. Specifically,*

$$\mathbb{P}(\overline{E^t_{i,j}}|\overline{E^t_{i,k}}) \geq p_s.$$

4. *Erasures in a neighbourhood are symmetric. Any subset of out-going edges of vertex i, will be erased with exactly the same probability as another subset of the same cardinality.*

The main two edge erasure models we consider are described here. They both satisfy all required properties. Our theory holds for both[1], but in our implementation and experiments we use "At Least One Out-Edge Per Node."

Example 28 (Independent Erasures)**.** *Every edge is preserved independently with probability p_s.*

[1] *Independent Erasures* can lose some walkers, when it temporarily leads to some nodes having zero out-degree.

Example 29 (At Least One Out-Edge Per Node)**.** *This edge erasure model, decides all erasures for node i independently, like* Independent Erasures*, but if all out-going edges for node i are erased, it draws and enables one of them uniformly at random.*

5.6.2 Proof of Theorem 24

In this section we provide a complete proof of our main results. We start from simple processes and slowly introduce the analytical intricacies of our system one-by-one giving guarantees on the performance of each stage.

Process 30 (Fixed Step)**.** *Independent walkers start on nodes selected uniformly at random and perform random walks on the* augmented *graph. This means that teleportation happens with probability p_T and the walk is described by the transition probability matrix (TPM) Q, as defined in Section 2.6. Each walker performs exactly t steps before yielding a sample. The number of walkers tends to infinity.*

Before we talk about the convergence properties of this Markov chain, we need some definitions.

Definition 31. *The χ^2-contrast $\chi^2(\alpha;\beta)$ of α with respect to β is defined by*

$$\chi^2(\alpha;\beta) = \sum_i \frac{(\alpha(i)-\beta(i))^2}{\beta(i)}.$$

Lemma 32. *Let $\pi \in \Delta^{n-1}$ a distribution satisfying $\min_i \pi(i) \geq \frac{c}{n}$ for constant $c \leq 1$, and let $u \in \Delta^{n-1}$ denote the uniform distribution. Then, $\chi^2(u;\pi) \leq \left(\frac{1-c}{c}\right)$.*

Proof.

$$\chi^2(u;\pi) = \sum_i \frac{(1/n - \pi(i))^2}{\pi(i)} = \sum_i \left(\frac{1}{n^2\pi(i)} - \frac{1}{n}\right)$$
$$= \frac{1}{n}\sum_i \frac{1 - n\pi(i)}{n\pi(i)} \leq \frac{1}{n}\sum_i \frac{1-c}{c} = \frac{1-c}{c}$$

Here we used the assumed lower bound on $\pi(i)$ and the fact that $(1-x)/x$ is decreasing in x. □

Lemma 33. *Let π^t denote the distribution of the walkers after t steps. Its χ^2-divergence with respect to the PageRank vector, π, is*

$$\chi^2(\pi^t;\pi) \leq \left(\frac{1-p_T}{p_T}\right)(1-p_T)^t.$$

Proof. Since Q is ergodic (but non-reversible) we can use the contrast bound in [20], which gives us

$$\chi^2(\pi^t;\pi) \leq \lambda_2^t(\tilde{Q}Q)\chi^2(\pi_0;\pi),$$

where $\tilde{Q} = DQ'D^{-1}$, for $D = diag(\pi)$, is called the *multiplicative reversibilization* of Q. We want an upper bound on the second largest eigenvalue of $\tilde{Q}Q = DQ'D^{-1}Q$. From the Perron-Frobenius theorem, we know that $\lambda_1(Q) = 1$ and from [33, 27, 66], $|\lambda_2(Q)| < 1 - p_T$. Matrix Q is similar to $\tilde{Q} = DQ'D^{-1}$, so they have the same spectrum. From this we get the bound

$$|\lambda_2(\tilde{Q}Q)| \leq 1 - p_T.$$

The starting distribution π_0 is assumed to be uniform and every element of the PageRank vector is lower bounded by p_T/n. From Lemma 32, we get

$$\chi^2(\pi_0;\pi) \leq \left(\frac{1-p_T}{p_T}\right),$$

and putting everything together we get the statement. □

Process 34 (Truncated Geometric)**.** *Independent walkers start on nodes selected uniformly at random and perform random walks on the* original *graph. This means that there is no teleportation and the walk is described by the TPM P as defined in Section 2.6. Each walker performs a random number of steps before yielding a sample. Specifically, the number of steps follows a geometric distribution with parameter p_T. Any walkers still active after t steps are stopped and their positions are acquired as samples. This means that the number of steps is given by the minimum of t and a geometric random variable with parameter p_T. The number of walkers tends to infinity.*

Lemma 35. *The samples acquired from Process 30 and Process 34 follow the exact same distribution.*

Proof. Let π_t denote the distribution of the walk after t steps according to Q (Process 30) and π'_t denote the distribution of the samples provided by the truncated geometric process (Process 34). Note that both have the same uniform starting distribution $\pi_0 = \pi'_0 = u = 1_{n\times 1}/n$. For the latter process, the sampling distribution is

$$\pi'_t = \sum_{\tau=0}^{t} p_T(1-p_T)^\tau P^\tau u + (1-p_T)^{t+1} P^t u. \tag{5.4}$$

The last term corresponds to the cut-off we impose at time t. Now consider the definition of the TPM Q (Definition 1). The Markov chain described by Q, teleports at each step with probability p_T; otherwise, it just proceeds according to the TPM P. With every teleportation, the walker starts from the

uniform distribution, u – any progress made so far is completely "forgotten." Therefore, we just need to focus on the epoch between the last teleportation and the cut-off time t. The times between teleportation events are geometrically distributed with parameter p_T. The teleportation process is memory-less and reversible. Starting from time t and looking backwards in time, the last teleportation event is a geometric number of steps away, and with probability $(1-p_T)^{t+1}$ it happens before the starting time 0. In that case we know that no teleportation happens in $[0,t]$. The samples acquired from this process are given by

$$\pi_t = \sum_{\tau=0}^{t} p_T(1-p_T)^\tau P^\tau u + (1-p_T)^{t+1} P^t u, \tag{5.5}$$

which is exactly the distribution for Process 34 given in (5.4). □

Lemma 36 (Mixing Loss)**.** *Let $\pi^t \in \Delta^{n-1}$ denote the distribution of the samples acquired through Process 34. The mass it captures (Definition 20) is lower-bounded as follows.*

$$\mu_k(\pi^t) \geq \mu_k(\pi) - \sqrt{\frac{(1-p_T)^{t+1}}{p_T}}$$

Proof. Let us define $\delta_i = \pi_i^t - \pi_i$. First we show that

$$\mu_k(\pi^t) \geq \mu_k(\pi) - \|\pi - \pi^t\|_1. \tag{5.6}$$

To see this, first consider the case when $\delta_1 = -\delta_2$ and $\delta_i = 0$ for $i = 3, ..., n$. The maximum amount of mass that can be missed by π^t, in this case, is $|\delta_1| + |\delta_2|$. This happens when π_1 and π_2 are exactly $|\delta_1| + |\delta_2|$ apart and are flipped in the ordering by π^t. This argument generalizes to give us (5.6). Now

assume that the χ^2-divergence of π^t with respect to the PageRank vector π is bounded by ϵ^2. Now using a variational argument and the KKT conditions we can show that setting $\delta_i = \pi_i \epsilon$ for all i gives the maximum possible l_1 error:

$$||\pi - \pi^t||_1 \leq \epsilon = \sqrt{\chi^2(\pi^t; \pi)}. \tag{5.7}$$

For another proof using the Cauchy-Schwarz inequality, see [20]. Finally, combining (5.7) with (5.6) and the results from Lemma 35 and 33 gives us the statement. □

Lemma 37 (Sampling Loss)**.** *Let $\hat{\pi}_N$ be the estimator of Definition 23 using N samples from the* FrogWild! *system. This is essentially, Process 34 with the added complication of* random synchronization *as explained in Section 5.2. Also, let π^t denote the sample distribution after t steps, as defined in Lemma 35. The mass captured by this process is lower bounded as follows, with probability at least $1 - \delta$.*

$$\mu_k(\hat{\pi}_N) \geq \mu_k(\pi^t) - \sqrt{\frac{k}{\delta}\left[\frac{1}{N} + (1 - p_s^2)p_\cap(t)\right]},$$

Proof. In this proof, let x_l^t denote the individual (marginal) walk distribution for walker l at time t. We know, that it follows the dynamics $x_l^{t+1} = Px_l^t$, for all $l \in [N]$, i.e. $x_l^t = x_1^t$. First we show that $\|\hat{\pi}_N - x_1^t\|_2$ is small.

$$\mathbb{P}(\|\hat{\pi}_N - x_1^t\|_2 > \epsilon) \leq \frac{\mathbb{E}[\|\hat{\pi}_N - x_1^t\|_2^2]}{\epsilon^2} \tag{5.8}$$

Here we used Markov's inequality. We use s_l^t to denote the position of walker l at time t as a vector. For example, $s_l^t = e_i$, if walker l is at state i at time t.

Now let us break down the norm on the numerator of (5.8).

$$\begin{aligned}\|\hat{\pi}_N - x_1^t\|_2^2 &= \left\| \frac{1}{N} \sum_l (s_l^t - x_1^t) \right\|_2^2 \\ &= \frac{1}{N^2} \sum_l \|s_l^t - x_1^t\|_2^2 + \frac{1}{N^2} \sum_{l \neq k} (s_l^t - x_1^t)'(s_k^t - x_1^t) \end{aligned} \tag{5.9}$$

For the diagonal terms we have:

$$\begin{aligned}\mathbb{E}[\|s_l^t - x_1^t\|_2^2] &= \sum_{i \in [n]} \mathbb{E}\left[\|s_l^t - x_1^t\|_2^2 | s_l^t = i\right] \mathbb{P}(s_l^t = i) \\ &= \sum_{i \in [n]} \|e_i^t - x_1^t\|_2^2 x_1^t(i) = 1 - \|x_1^t\|_2^2 \leq 1 \end{aligned} \tag{5.10}$$

Under the edge erasures model, the trajectories of different walkers are not generally independent. For example, if they happen to meet, they are likely to make the same decision for their next step, since they are faced with the same edge erasures. Now we prove that even when they meet, we can consider them to be independent with some probability that depends on p_s.

Consider the position processes for two walkers, $\{s_1^t\}_t$ and $\{s_2^t\}_t$. At each step t and node i a number of out-going edges are erased. Any walkers on i, will choose uniformly at random from the remaining edges. Now consider this *alternative process.*

Process 38 (Blocking Walk)**.** *A blocking walk on the graph under the erasure model, follows these steps.*

1. *Walker l finds herself on node i at time t.*

2. *Walker l draws her next state uniformly from the* full *set of out-going edges.*

$$w \sim \text{Uniform}(\mathcal{N}_o(i))$$

3. *If the edge (i, w) is erased at time t, the walker cannot traverse it. We call this event a* block *and denote it by B_l^t. In the event of a block:*

   - *Walker redraws her next step from the out-going edges of i* not erased *at time t.*
   - *Otherwise, w is used as the next state.*

A blocking walk is exactly equivalent to our original process; walkers end up picking a destination uniformly at random among the edges not erased. From now on we focus on this description of our original process. We use the same notation: $\{s_l^t\}_t$ for the position process and $\{x_l^t\}_t$ for the distribution at time t.

Let us focus on just two walkers, $\{s_1^t\}_t$ and $\{s_2^t\}_t$ and consider a third process: two independent random walks on the same graph. We assume that these walks operate on the full graph, i.e. no edges are erased. We denote their positions by $\{v_1^t\}_t$ and $\{v_2^t\}_t$ and their marginal distributions by $\{z_1^t\}_t$ and $\{z_2^t\}_t$.

Definition 39 (Time of First Interference)**.** *For two blocking walks, τ_I denotes the earliest time at which they meet and at least one of them experiences blocking.*

$$\tau_I = \min\left\{t : \{s_1^t = s_2^t\} \cap (B_1^t \cup B_2^t)\right\}$$

We call this quantity the time of first interference.

Lemma 40 (Process equivalence)**.** *For two walkers, the blocking walk and the independent walk are identical until the time of first interference. That is, assuming the same starting distributions,* $x_1^0 = z_1^0$ *and* $x_2^0 = z_2^0$*, then*

$$x_1^t = z_1^t \quad \textit{and} \quad x_2^t = z_2^t \qquad \forall t \leq \tau_I.$$

Proof. The two processes are equivalent for as long as the blocking walkers make *independent* decisions effectively picking *uniformly from the full set of edges* (before erasures). From the independence in erasures across time and vertices in Definition 27, as long as the two walkers do not meet, they are making an independent choices. Furthermore, since erasures are symmetric, the walkers will be effectively choosing uniformly over the full set of out-going edges.

Now consider any time t that the blocking walkers meet. As long as neither of them blocks, they are by definition taking independent steps uniformly over the set of all outgoing edges, maintaining equivalence to the independent walks process. This concludes the proof. □

Lemma 41. *Let all walkers start from the uniform distribution. The probability that the time of first interference comes before time* t *is upper bounded as follows.* $\mathbb{P}(\tau_I \leq t) \leq (1 - p_s^2)p_\cap(t)$

Proof. Let M_t be the event of a meeting at time t, $M_t \triangleq \{s_1^t = s_2^t\}$. In the proof of Theorem 25, we establish that $\mathbb{P}(M_t) \leq \rho^t/n$,where ρ is the maximum

row sum of the transition matrix P. Now denote the event of an interference at time t as follows. $I_t \triangleq M_t \cap (B_1^t \cup B_2^t)$, where B_1^t denotes the event of blocking, as described in Definition 38. Now,

$$\mathbb{P}(I_t) = \mathbb{P}(M_t \cap (B_1^t \cup B_2^t)) = \mathbb{P}(B_1^t \cup B_2^t | M_t) p_{\cap}(t).$$

For the probability of a block given that the walkers meet at time t,

$$\begin{aligned}\mathbb{P}(B_1^t \cup B_2^t | M_t) &= 1 - \mathbb{P}(\overline{B_1^t} \cap \overline{B_2^t} | M_t) \\ &= 1 - \mathbb{P}(\overline{B_2^t} | \overline{B_1^t}, M_t)\mathbb{P}(\overline{B_1^t} \,| M_t) \leq 1 - p_s^2.\end{aligned}$$

To get the last inequality we used, from Definition 27, the lower bound on the probability that an edge is not erased, and the lack of negative correlations in the erasures.

Combining the above results, we get

$$\begin{aligned}\mathbb{P}(\tau_I \leq t) = \mathbb{P}\left(\sum_{\tau=1}^{t} \mathbb{I}_{\{I_\tau\}} \geq 1\right) &\leq \mathbb{E}\left[\sum_{\tau=1}^{t} \mathbb{I}_{\{I_\tau\}}\right] = \sum_{\tau=1}^{t} \mathbb{P}(I_\tau) \\ \leq \sum_{\tau=1}^{t} (1 - p_s^2)\mathbb{P}(M_\tau) = \frac{1 - p_s^2}{n} \sum_{\tau=1}^{t} \rho^\tau &= (1 - p_s^2) p_{\cap}(t)\end{aligned}$$

which proves the statement. □

Now we can bound the off-diagonal terms in (5.10).

$$\begin{aligned}&\mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\Big] \\ &\qquad = \mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\Big|\tau_I \leq t\Big]\mathbb{P}(\tau_I \leq t) \\ &\qquad\quad + \mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\Big|\tau_I > t\Big]\mathbb{P}(\tau_I > t)\end{aligned}$$

In the second term, the case when l, k have not interfered, by Lemma 40, the trajectories are independent and the cross-covariance is 0. In the first term, the cross-covariance is maximized when $s_l^t = s_k^t$. That is,

$$\mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\big|\tau_I \le t\Big] \le \mathbb{E}[\|s_l^t - x_1^t\|_2^2] \le 1$$

From this we get

$$\mathbb{E}\Big[(s_l^t - x_1^t)'(s_k^t - x_1^t)\Big] \le (1 - p_s^2)p_\cap(t), \tag{5.11}$$

and in combination with (5.10), we get from (5.9) that

$$\mathbb{E}\Big[\|\hat{\pi}_N - x_1^t\|_2^2\Big] \le \frac{1}{N} + \frac{(N-1)(1-p_s^2)p_\cap(t)}{N}.$$

Finally, we can plug this into (5.8), and since all marginals x_l^t are the same, and denoted by π^t, we get

$$\mathbb{P}(\|\hat{\pi}_N - \pi^t\|_2 > \epsilon) \le \frac{1 + (1-p_s^2)p_\cap(t)(N-1)}{N\epsilon^2}. \tag{5.12}$$

Let $\pi^t|_S$ denote the restriction of the vector π^t to the set S. That is, $\pi^t|_S(i) = \pi^t(i)$ if $i \in S$ and 0 otherwise. Now we show that for any set S of cardinality k,

$$\begin{aligned} |\pi^t(S) - \hat{\pi}_N(S)| &\le \|(\pi^t - \hat{\pi}_N)\big|_S\|_1 \le \sqrt{k}\|(\pi^t - \hat{\pi}_N)\big|_S\|_2 \\ &\le \sqrt{k}\|\pi^t - \hat{\pi}_N\|_2 \end{aligned} \tag{5.13}$$

Here we used the fact that for k-length vector x, $\|x\|_1 \le \sqrt{k}\|x\|_2$ and $\|x|_S\| \le \|x\|$. We define the top-k sets

$$\hat{S}^* = \text{argmax}_{S\subset[n],|S|=k}\hat{\pi}_N(S)$$

$$S^* = \text{argmax}_{S\subset[n],|S|=k}\pi^t(S).$$

Per these definitions,

$$\begin{aligned}\hat{\pi}_N(\hat{S}^*) &= \max_{S\subset[n],|S|=k}\hat{\pi}_N(S) \\ &\geq \hat{\pi}_N(S^*) \geq \pi^t(S^*) - \sqrt{k}\|\pi^t - \hat{\pi}_N\|_2. \end{aligned} \tag{5.14}$$

The last inequality is a consequence of (5.13). Now using the inequality in (5.12) and denoting the LHS probability as δ, we get the statement of Lemma 37. □

Combing the results of Lemma 36 and Lemma 37, we establish the main result, Theorem 24.

5.6.3 Proof of Theorem 25

Proof. Let $u \in \Delta^{n-1}$ denote the uniform distribution over $[n]$, i.e. $u_i = 1/n$. The two walks start from the same initial uniform distribution, u, and independently follow the same law, Q. Hence, at time t they have the same marginal distribution, $p^t = Q^t u$. From the definition of the augmented transition probability matrix, Q, in Definition 1, we get that

$$\pi_i \geq \frac{p_T}{n}, \quad \forall i \in [n].$$

Equivalently, there exists a distribution $q \in \Delta^{n-1}$ such that

$$\pi = p_T u + (1 - p_T)q.$$

Now using this, along with the fact that π is the invariant distribution associated with Q (i.e. $\pi = Q^t\pi$ for all $t \geq 0$) we get that for any $t \geq 0$,

$$\begin{aligned}\|\pi\|_\infty &= \|Q^t\pi\|_\infty \\ &= \|Q^t p_T u + Q^t(1-p_T)q\|_\infty \\ &\geq p_T\|Q^t u\|_\infty.\end{aligned}$$

For the last inequality, we used the fact that Q and q contain non-negative entries. Now we have a useful upper bound for the maximal element of the walks' distribution at time t.

$$\|p^t\|_\infty = \|Q^t u\|_\infty \leq \frac{\|\pi\|_\infty}{p_T} \tag{5.15}$$

Let M_t be the indicator random variable for the event of a meeting at time t.

$$M_t = \mathbb{I}_{\{\text{walkers meet at time } t\}}$$

Then, $\mathbb{P}(M_t = 1) = \sum_{i=1}^n p_i^t p_i^t = \|p^t\|_2^2$. Since p^0 is the uniform distribution, i.e. $p_i^0 = \frac{1}{n}$ for all i, then $\|p^0\|_2^2 = \frac{1}{n}$. We can also bound the l_2 norm of the distribution at other times. First, we upper bound the l_2 norm by the l_∞ norm.

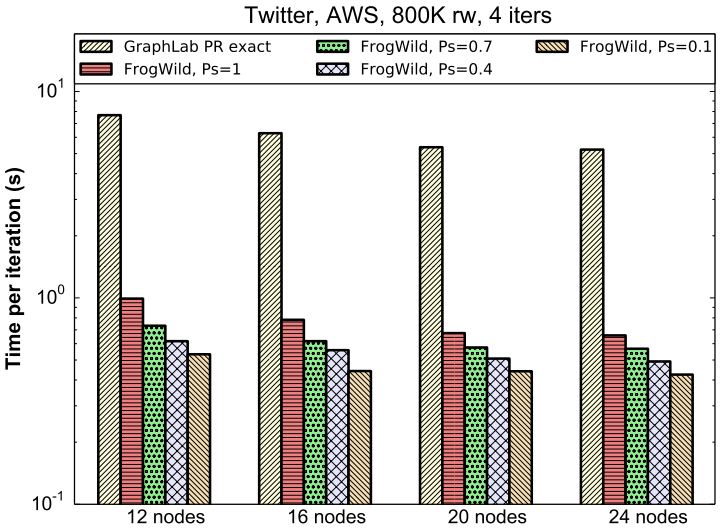
$$\|p\|_2^2 = \sum_i p_i^2 \leq \sum_i p_i\|p\|_\infty = \|p\|_\infty$$

Here we used the fact that $p_i \geq 0$ and $\sum p_i = 1$.

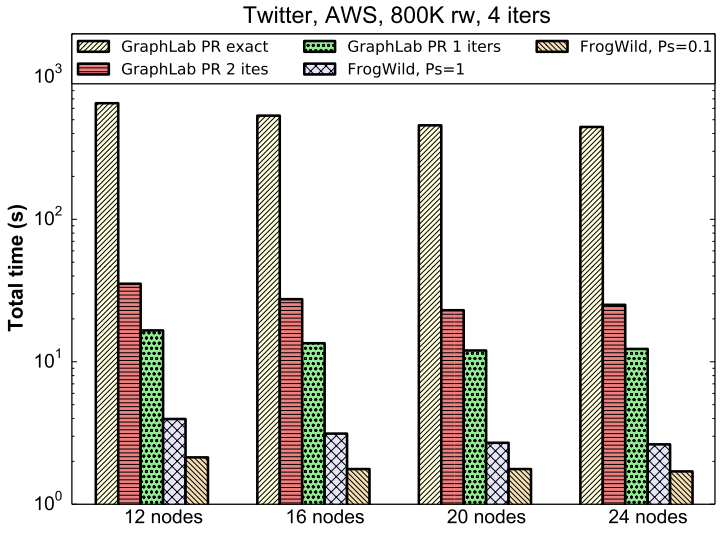
Now, combining the above results, we get

$$\begin{aligned} p_{\cap}(t) &= \mathbb{P}\left(\sum_{\tau=0}^{t} M_\tau \geq 1\right) \leq \mathbb{E}\left[\sum_{\tau=0}^{t} M_\tau\right] = \sum_{\tau=0}^{t} \mathbb{E}[M_\tau] \\ &= \sum_{\tau=0}^{t} \mathbb{P}(M_\tau = 1) = \sum_{\tau=0}^{t} \|p^\tau\|_2^2 \leq \sum_{\tau=0}^{t} \|p^\tau\|_\infty \\ &\leq \frac{1}{n} + \frac{t\|\pi\|_\infty}{p_T}. \end{aligned}$$

For the last inequality, we used (5.15) for $t \geq 1$ and $\|p^0\|_2^2 = 1/n$. This proves the theorem statement. □

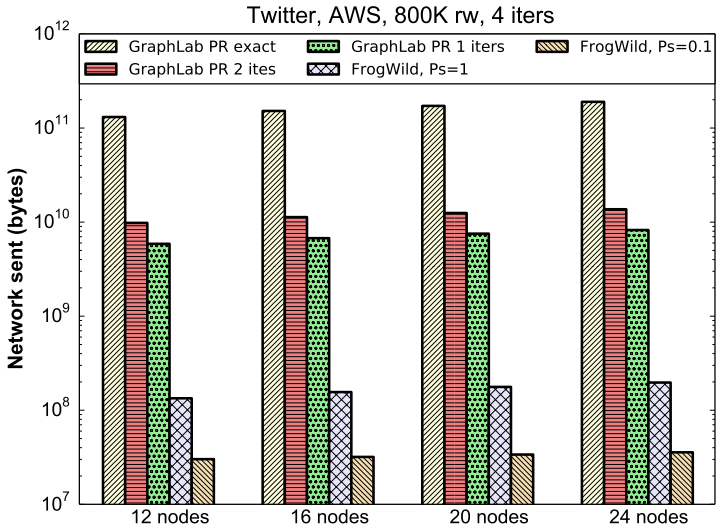


(a)

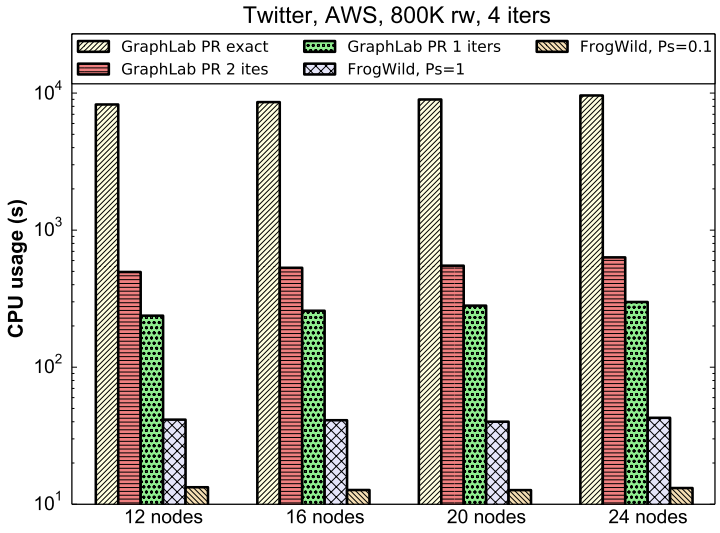


(b)

Figure 5.1: PageRank performance for various number of nodes. Graph: Twitter; system: AWS (Amazon Web Services); FrogWild parameters: 800K initial random walks and 4 iterations. (a) Running time per iteration. (b) Total running time of the algorithms.

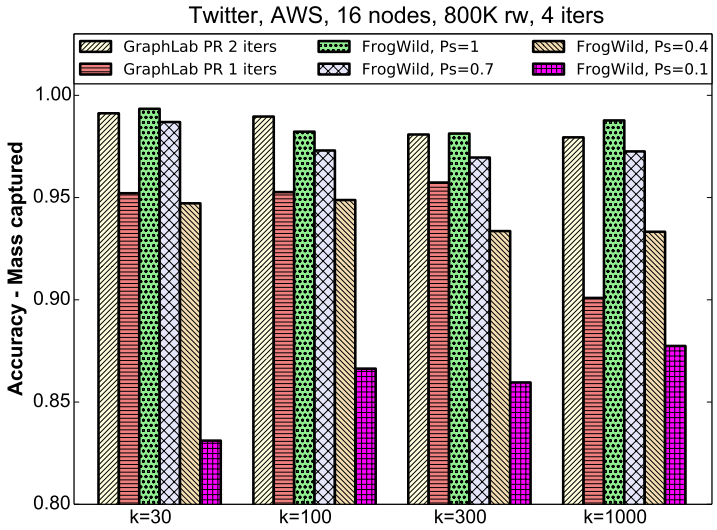


(c)

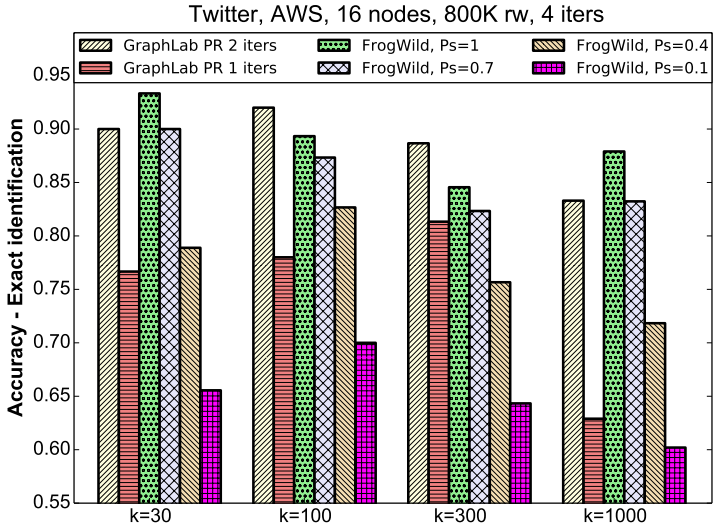


(d)

Figure 5.2: PageRank performance for various number of nodes. Graph: Twitter; system: AWS (Amazon Web Services); FrogWild parameters: 800K initial random walks and 4 iterations. (c) Total network bytes sent by the algorithm during the execution (does not include ingress time). (d) Total CPU usage time. Notice, this metric may be larger than the total running time since many CPUs run in parallel.

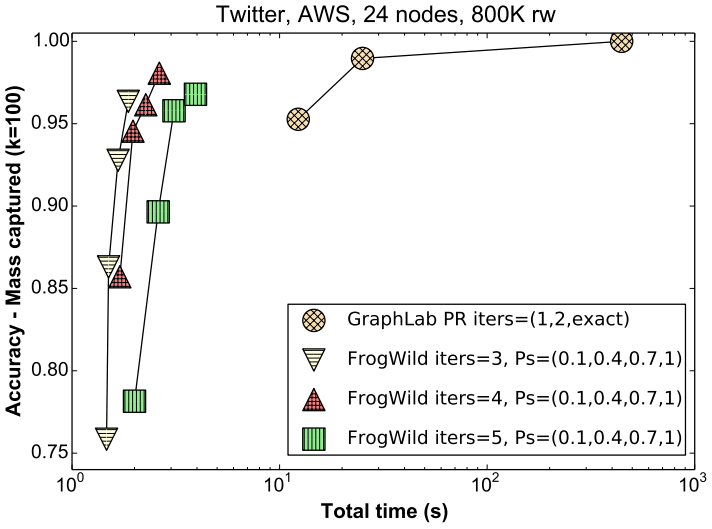


(a)

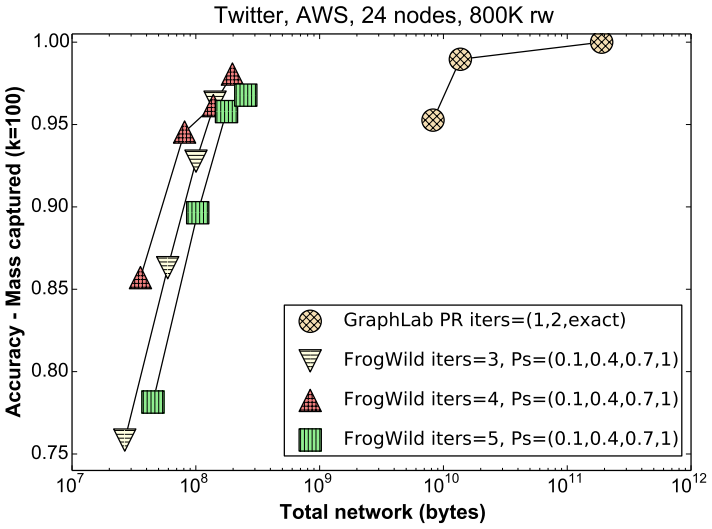


(b)

Figure 5.3: PageRank approximation accuracy for various number of top-k PageRank vertices. Graph: Twitter; system: AWS (Amazon Web Services) with 16 nodes; FrogWild parameters: 800K initial random walks and 4 iterations. (a) Mass captured. The total PageRank that the reported top-k vertices worth in the exact ranking. (b) Exact identification. The number of vertices in the intersection of the reported top-k and the exact top-k lists.



(a)



(b)

Figure 5.4: PageRank approximation accuracy with the "Mass captured" metric for top-100 vertices. Graph: Twitter; system: AWS (Amazon Web Services) with 24 nodes; FrogWild parameters: 800K initial random walks. (a) - Accuracy versus total running time. (b) - Accuracy versus total network bytes sent.

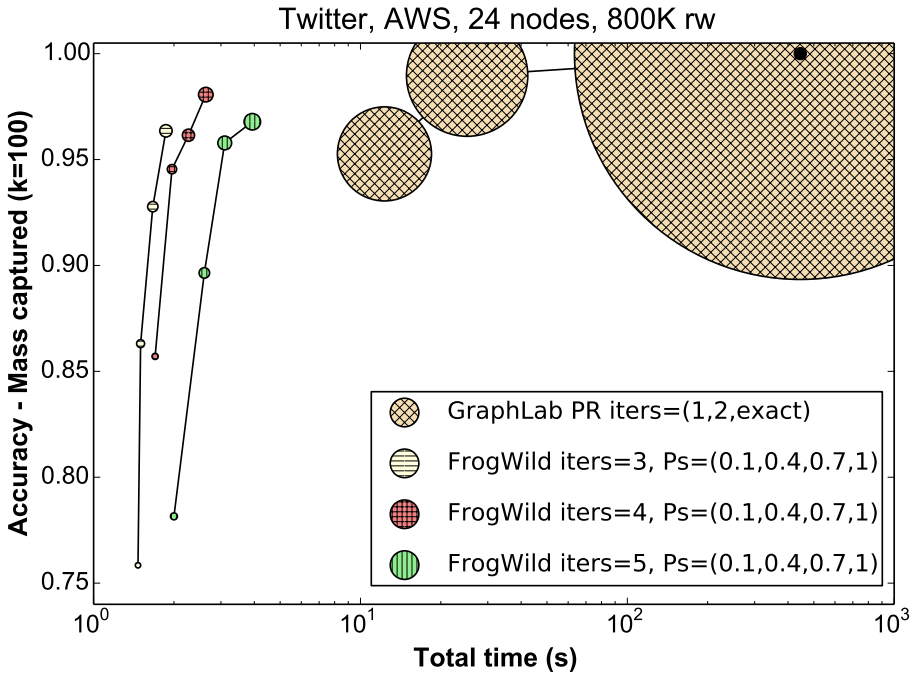


Figure 5.5: Accuracy versus total running time. Graph: Twitter; system: AWS (Amazon Web Services) with 24 nodes; FrogWild parameters: 800K initial random walks. The area of each circle is proportional to the total network bytes sent by the specific algorithm.

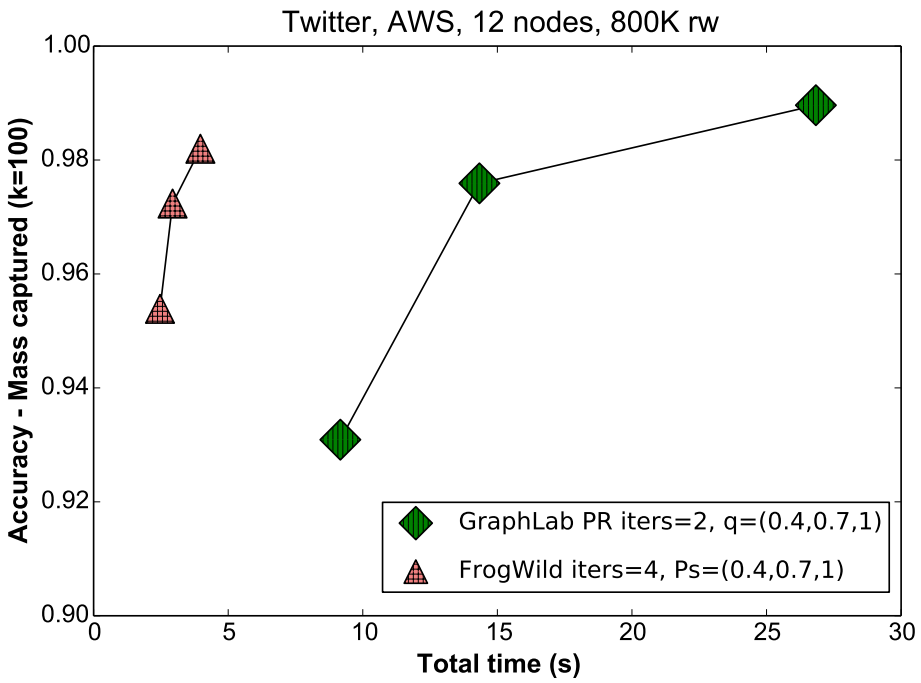
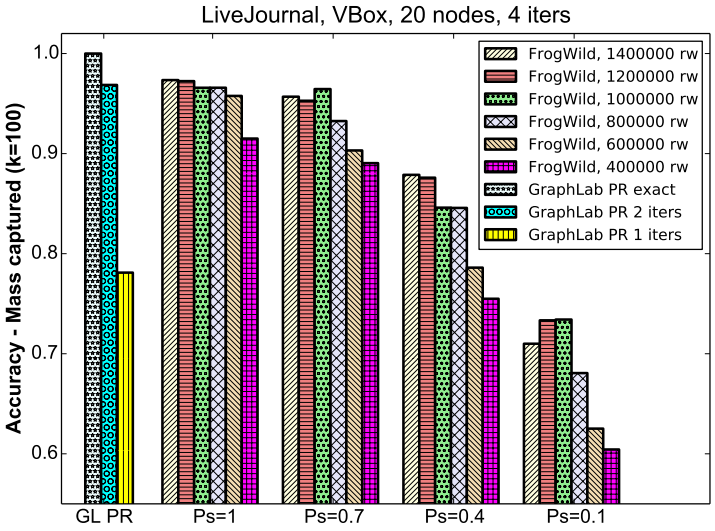
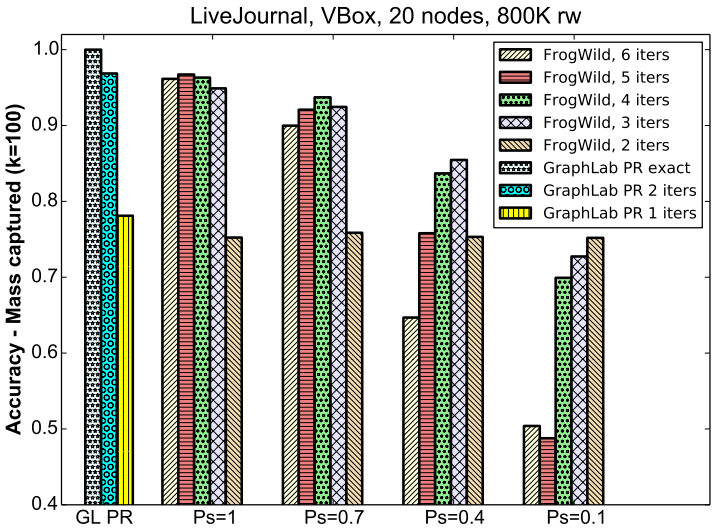


Figure 5.6: Accuracy versus total running time. Graph: Twitter; system: AWS (Amazon Web Services) with 12 nodes; FrogWild parameters: 800K initial random walks. $q = 1 - r$ is the probability of *keeping* an edge in the sparsification process.

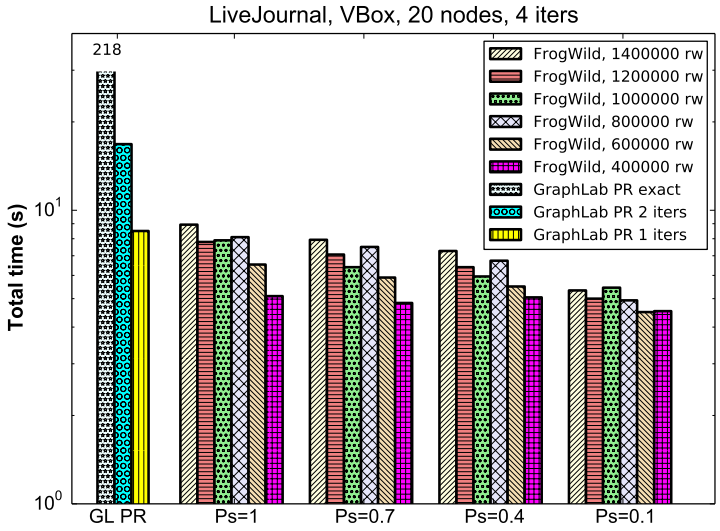


(a)

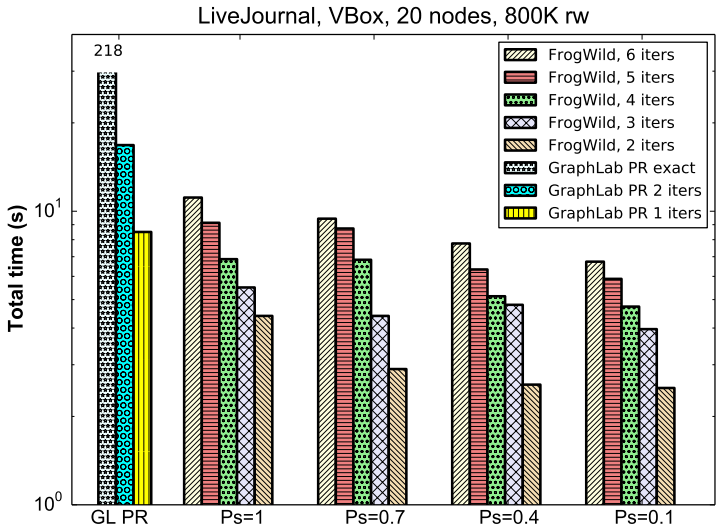


(b)

Figure 5.7: Graph: LiveJournal; system: VirtualBox with 20 nodes. (a) Accuracy for various number of initial random walks in the FrogWild (with 4 iterations). (b) Accuracy for various number of iterations of FrogWild (with 800K initial random walks).

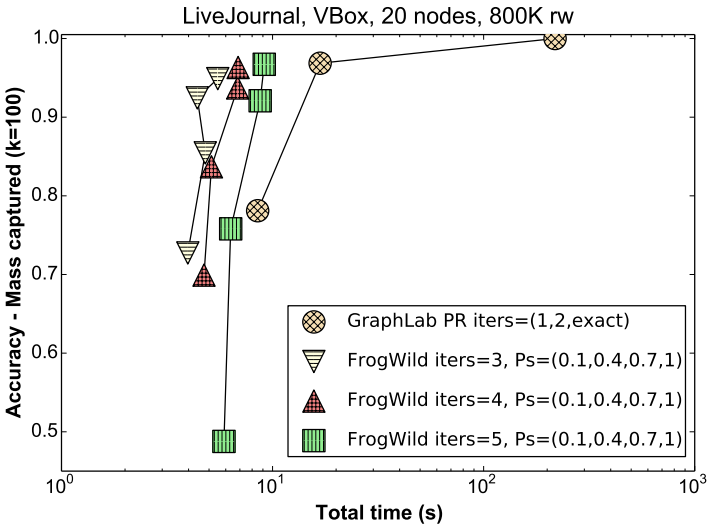


(c)

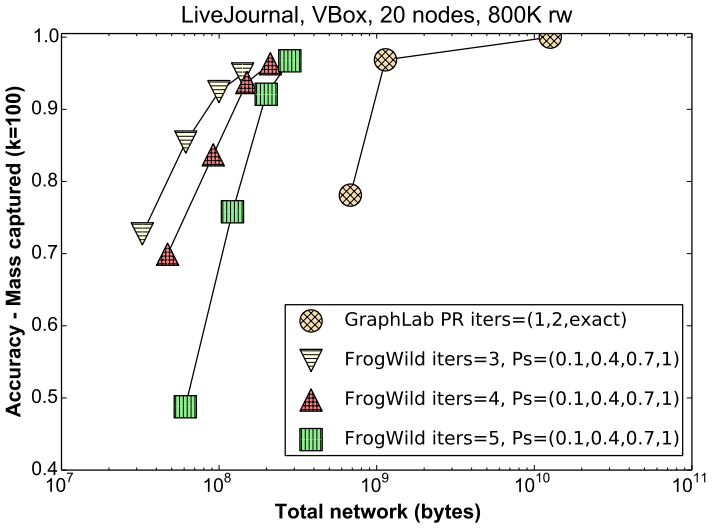


(d)

Figure 5.8: Graph: LiveJournal; system: VirtualBox with 20 nodes. (c) Total running time for various number of initial random walks in the FrogWild (with 4 iterations). (d) Total running time for various number of iterations of FrogWild (with 800K initial random walks).



(a)



(b)

Figure 5.9: Graph: LiveJournal; system: VirtualBox with 20 nodes; FrogWild parameters: 800K initial random walks. (a) Accuracy versus total running time. (b) Accuracy versus total network bytes sent.

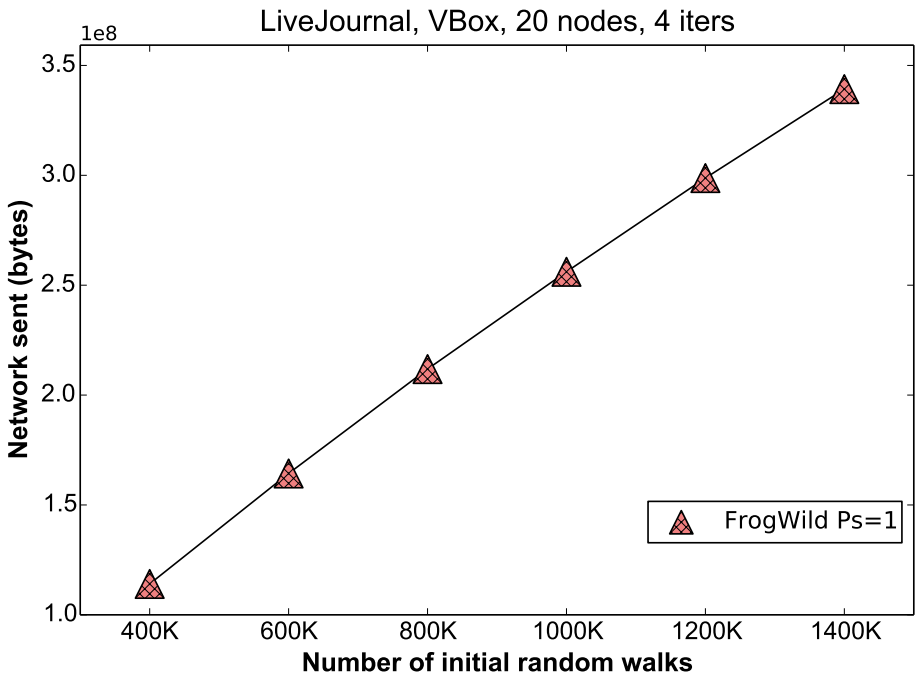


Figure 5.10: Network usage of FrogWild versus the number of initial random walks. Graph: LiveJournal; system: VirtualBox with 20 nodes; FrogWild parameters: 4 iterations.

Bibliography

[1] Teradata. `http://www.teradata.com/Resources/Videos/Grow-Loyalty-of-Influential-Customers`. Accessed: 2014-11-30. 9

[2] Alekh Agarwal and Soumen Chakrabarti. Learning random walks to rank nodes in graphs. In *Proceedings of the 24th international conference on Machine learning*, pages 9–16. ACM, 2007. 17

[3] Amazon web services. http://aws.amazon.com, 2014. 80

[4] Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcraft, Vahab S Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. In *Algorithms and Models for the Web-Graph*, pages 150–165. Springer, 2007. 77

[5] R. Arora, A. Cotter, K. Livescu, and N. Srebro. Stochastic optimization for PCA and PLS. In *50th Allerton Conference on Communication, Control, and Computing*, Monticello, IL, 2012. 12, 21, 22, 23

[6] Raman Arora, Andrew Cotter, Karen Livescu, and Nathan Srebro. Stochastic optimization for PCA and PLS. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 861–868. IEEE, 2012. 6

[7] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM J. Numer. Anal.*, 45(2):890–904, February 2007. 77

[8] Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, Elena Smirnova, and Marina Sokol. Monte carlo methods for top-k personalized pagerank lists and name disambiguation. *CoRR*, abs/1008.3775, 2010. 77

[9] Akshay Balsubramani, Sanjoy Dasgupta, and Yoav Freund. The fast convergence of incremental PCA. In *Advances in Neural Information Processing Systems*, pages 3174–3182, 2013. 6, 22, 54

[10] L. Balzano, R. Nowak, and B. Recht. Online identification and tracking of subspaces from highly incomplete information. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, page 704–711, 2010. 21

[11] Laura Balzano, Robert Nowak, and Benjamin Recht. Online identification and tracking of subspaces from highly incomplete information. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 704–711. IEEE, 2010. 6, 13, 54

[12] Laura Balzano and Stephen J Wright. Local convergence of an algorithm for subspace identification from partial data. *arXiv preprint arXiv:1306.3391*, 2013. 6, 13, 47

[13] Joshua Batson, Daniel A. Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: Theory and algorithms. *Commun. ACM*, 56(8):87–94, August 2013. 78

[14] Luca Becchetti and Carlos Castillo. The distribution of pagerank follows a power-law only for particular values of the damping factor. In *Proceedings of the 15th international conference on World Wide Web*, pages 941–942. ACM, 2006. 76

[15] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in Õ(n2) time. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 47–55, New York, NY, USA, 1996. ACM. 78

[16] Pavel Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1):73–120, 2005. 77

[17] Michael Borokhovich and Ioannis Mitliagkas. FrogWild! code repository. `https://github.com/michaelbor/frogwild`, 2014. Accessed: 2014-10-30. 82

[18] M. Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006. 21

[19] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. *Computer Vision—ECCV 2002*, page 707–720, 2002. 21

[20] Pierre Bremaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. springer, 1999. 68, 87, 90

[21] Andrei Z Broder, Ronny Lempel, Farzin Maghoul, and Jan Pedersen. Efficient pagerank approximation via graph aggregation. *Information Retrieval*, 9(2):123–138, 2006. 77

[22] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717–772, 2009. 5

[23] Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, page 205–214, 2009. 21

[24] P. Comon and G. H. Golub. Tracking a few extreme singular values and vectors in signal processing. *Proceedings of the IEEE*, 78(8):1327–1343, 1990. 21

[25] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *Journal of the ACM (JACM)*, 60(1):2, 2013. 65, 77

[26] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977. 23

[27] Lars Eldén. A note on the eigenvalues of the google matrix. *arXiv preprint math/0401177*, 2004. 87

[28] Santo Fortunato and Alessandro Flammini. Random walks on directed networks: the case of pagerank. *International Journal of Bifurcation and Chaos*, 17(07):2343–2353, 2007. 17

[29] Gene H. Golub and Charles F. Van Loan. *Matrix computations*, volume 3. JHUP, 2012. 3

[30] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2, 2012. 16, 66, 67, 79

[31] GroupLens. Movielens dataset. http://grouplens.org/datasets/movielens/, 2009. Accessed: 2014-02-21. 58

[32] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011. 21

[33] Taher Haveliwala and Sepandar Kamvar. The second eigenvalue of the google matrix. *Stanford University Technical Report*, 2003. 87

[34] J. He, L. Balzano, and J. Lui. Online robust subspace tracking from partial information. *arXiv preprint arXiv:1109.3827*, 2011. 21

[35] Jun He, Laura Balzano, and John Lui. Online robust subspace tracking from partial information. *arXiv preprint arXiv:1109.3827*, 2011. 13

[36] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005. 22

[37] Julia Heidemann, Mathias Klier, and Florian Probst. Identifying key users in online social networks: A pagerank based approach. In *ICIS'10*, 2010. 9, 10

[38] Mark Herbster and Manfred K. Warmuth. Tracking the best linear predictor. *The Journal of Machine Learning Research*, 1:281–309, 2001. 22

[39] Iain M. Johnstone. On the distribution of the largest eigenvalue in principal components analysis. *Ann. Statist*, 29(2):295–327, 2001. 3

[40] Iain M Johnstone. On the distribution of the largest eigenvalue in principal components analysis. *The Annals of statistics*, 29(2):295–327, 2001. 12, 22

[41] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from noisy entries. *Journal of Machine Learning Research*, 11(2057-2078):1, 2010. 5

[42] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM. 80

[43] Jason D Lee, Ben Recht, Ruslan Salakhutdinov, Nathan Srebro, and Joel A Tropp. Practical large-scale optimization for max-norm regularization. In *NIPS*, pages 1297–1305, 2010. 5

[44] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014. 80

[45] Y. Li. On incremental and robust subspace learning. *Pattern recognition*, 37(7):1509–1518, 2004. 21

[46] Peter Lofgren, Siddhartha Banerjee, Ashish Goel, and C Seshadhri. Fast-ppr: Scaling personalized pagerank estimation for large graphs. *arXiv preprint arXiv:1404.3181*, 2014. 77

[47] Karim Lounici. High-dimensional covariance matrix estimation with missing observations. *arXiv preprint arXiv:1201.2577*, 2012. 13, 51, 52

[48] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, July 2010. 8, 15, 79

[49] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010. 8, 15

[50] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. Association for Computational Linguistics, 2004. 9

[51] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. Memory limited, streaming PCA. *arXiv preprint arXiv:1307.0032*, 2013. 6

[52] Boaz Nadler. Finite sample approximation results for principal component analysis: a matrix perturbation approach. *The Annals of Statistics*, page 2791–2817, 2008. 21

[53] Sahand Negahban, Martin J Wainwright, et al. Estimation of (near) low-rank matrices with noise and high-dimensional scaling. *The Annals of Statistics*, 39(2):1069–1097, 2011. 5

[54] Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005. 76

[55] Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications*, 106(1):69–84, 1985. 6, 53, 54

[56] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999. 8, 17, 71

[57] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, pages 77–85, New York, NY, USA, 1987. ACM. 78

[58] Ian Porteous, David Newman, Alexander Ihler, Arthur Asuncion, Padhraic Smyth, and Max Welling. Fast collapsed gibbs sampling for latent dirichlet allocation. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, page 569–577, 2008. 45

[59] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011. 67

[60] Jasson DM Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005. 5

[61] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, page 400–407, 1951. 22

[62] Sam Roweis. EM algorithms for PCA and SPCA. *Advances in neural information processing systems*, page 626–632, 1998. 22

[63] Mark Rudelson and Roman Vershynin. Smallest singular value of a random rectangular matrix. *Communications on Pure and Applied Mathematics*, 62(12):1707–1739, 2009. 32, 42

[64] Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Estimating pagerank on graph streams. *Journal of the ACM (JACM)*, 58(3):13, 2011. 65, 77

[65] Nadathur Satish, Narayanan Sundaram, Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey. Navigating the maze of graph analytics frameworks using massive graph datasets. 8, 15, 79

[66] Stefano Serra-Capizzano. Jordan canonical form of the google matrix: a potential contribution to the pagerank computation. *SIAM Journal on Matrix Analysis and Applications*, 27(2):305–312, 2005. 87

[67] D. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. 78

[68] Michael E. Tipping and Christopher M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999. 22

[69] Joel A Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434, 2012. 52

[70] Leslie G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990. 16

[71] Alexander Vergara, Shankar Vembu, Tuba Ayhan, Margaret A Ryan, Margie L Homer, and Ramón Huerta. Chemical gas sensor drift com-

pensation using classifier ensembles. *Sensors and Actuators B: Chemical*, 166:320–329, 2012. 56

[72] R. Vershynin. How close is the sample covariance matrix to the actual covariance matrix? *Journal of Theoretical Probability*, page 1–32, 2010. 3, 32

[73] Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*, 2010. 20, 32, 33

[74] VirtualBox 4.3. www.virtualbox.org, 2014. 80

[75] Manfred K. Warmuth and Dima Kuzmin. Randomized online PCA algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9:2287–2320, 2008. 12, 20

[76] Per Åke Wedin. On angles between subspaces of a finite dimensional inner product space. In *Matrix Pencils*, pages 263–285. Springer, 1983. 49

[77] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2013. 8, 15

Vita

Ioannis Mitliagkas was born in Kozani, Greece on April 13 1984, the son of Vasilis Mitliagkas and Despoina Tzika. He received his 5 year diploma and M.Sc. in Electronic and Computer Engineering from the Technical University of Crete, with an award of excellence by the Technical Chamber of Greece. He was admitted as a fellow into the ECE PhD program at UT Austin in 2009. As of Fall 2015 he is a postdoctoral scholar with the Department of Statistics at Stanford University.

Permanent address: Methonis 41
Kozani, 50100
Greece

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.